

# The OpenXR™ 1.1.36 Specification (with all registered extensions)

The Khronos® OpenXR Working Group

Version 1.1.36: from git ref release-1.1.36

# Table of Contents

|  |    |
|--|----|
| Preamble                               | 1  |
| 1. Introduction                        | 3  |
| 1.1. What is OpenXR?                   | 3  |
| 1.2. The Programmer's View of OpenXR   | 3  |
| 1.3. The Implementor's View of OpenXR  | 3  |
| 1.4. Our View of OpenXR                | 4  |
| 1.5. Filing Bug Reports                | 4  |
| 1.6. Document Conventions              | 4  |
| 2. Fundamentals                        | 6  |
| 2.1. API Version Numbers and Semantics | 6  |
| 2.2. String Encoding                   | 8  |
| 2.3. Threading Behavior                | 8  |
| 2.4. Multiprocessing Behavior          | 10 |
| 2.5. Runtime                           | 10 |
| 2.6. Extensions                        | 10 |
| 2.7. API Layers                        | 11 |
| 2.8. Type Aliasing                     | 13 |
| 2.9. Valid Usage                       | 14 |
| 2.10. Return Codes                     | 18 |
| 2.11. Handles                          | 31 |
| 2.12. Object Handle Types              | 32 |
| 2.13. Buffer Size Parameters           | 34 |
| 2.14. Time                             | 36 |
| 2.15. Duration                         | 37 |
| 2.16. Prediction Time Limits           | 38 |
| 2.17. Colors                           | 38 |
| 2.18. Coordinate System                | 39 |
| 2.19. Common Data Types                | 42 |
| 2.20. Angles                           | 48 |
| 2.21. Boolean Values                   | 49 |
| 2.22. Events                           | 50 |
| 2.23. System resource lifetime         | 55 |
| 3. API Initialization                  | 56 |
| 3.1. Exported Functions                | 56 |
| 3.2. Function Pointers                 | 56 |
| 3.3. Runtime Interface Negotiation     | 59 |

|  |     |
|--|-----|
| 3.4. API Layer Interface Negotiation           | 65  |
| 4. Instance                                    | 74  |
| 4.1. API Layers and Extensions                 | 74  |
| 4.2. Instance Lifecycle                        | 79  |
| 4.3. Instance Information                      | 85  |
| 4.4. Platform-Specific Instance Creation       | 87  |
| 4.5. Instance Enumerated Type String Functions | 88  |
| 5. System                                      | 91  |
| 5.1. Form Factors                              | 91  |
| 5.2. Getting the XrSystemId                    | 92  |
| 5.3. System Properties                         | 95  |
| 6. Path Tree and Semantic Paths                | 99  |
| 6.1. Path Atom Type                            | 99  |
| 6.2. Well-Formed Path Strings                  | 101 |
| 6.3. Reserved Paths                            | 105 |
| 6.4. Interaction Profile Paths                 | 112 |
| 7. Spaces                                      | 144 |
| 7.1. Reference Spaces                          | 145 |
| 7.2. Action Spaces                             | 151 |
| 7.3. Space Lifecycle                           | 152 |
| 7.4. Locating Spaces                           | 159 |
| 8. View Configurations                         | 174 |
| 8.1. Primary View Configurations               | 174 |
| 8.2. View Configuration API                    | 176 |
| 8.3. Example View Configuration Code           | 183 |
| 9. Session                                     | 186 |
| 9.1. Session Lifecycle                         | 186 |
| 9.2. Session Creation                          | 188 |
| 9.3. Session Control                           | 192 |
| 9.4. Session States                            | 197 |
| 10. Rendering                                  | 202 |
| 10.1. Swapchain Image Management               | 202 |
| 10.2. View and Projection State                | 220 |
| 10.3. Frame Synchronization                    | 225 |
| 10.4. Frame Submission                         | 229 |
| 10.5. Frame Rate                               | 235 |
| 10.6. Compositing                              | 236 |
| 11. Input and Haptics                          | 249 |

|  |     |
|--|-----|
| 11.1. Action Overview                                | 249 |
| 11.2. Action Sets                                    | 251 |
| 11.3. Creating Actions                               | 255 |
| 11.4. Suggested Bindings                             | 260 |
| 11.5. Current Interaction Profile                    | 266 |
| 11.6. Reading Input Action State                     | 270 |
| 11.7. Output Actions and Haptics                     | 282 |
| 11.8. Input Action State Synchronization             | 287 |
| 11.9. Bound Sources                                  | 290 |
| 12. List of Current Extensions                       | 297 |
| 12.1. XR_KHR_android_create_instance                 | 302 |
| 12.2. XR_KHR_android_surface_swapchain               | 304 |
| 12.3. XR_KHR_android_thread_settings                 | 307 |
| 12.4. XR_KHR_binding_modification                    | 311 |
| 12.5. XR_KHR_composition_layer_color_scale_bias      | 314 |
| 12.6. XR_KHR_composition_layer_cube                  | 317 |
| 12.7. XR_KHR_composition_layer_cylinder              | 320 |
| 12.8. XR_KHR_composition_layer_depth                 | 324 |
| 12.9. XR_KHR_composition_layer_equirect              | 328 |
| 12.10. XR_KHR_composition_layer_equirect2            | 332 |
| 12.11. XR_KHR_convert_timespec_time                  | 335 |
| 12.12. XR_KHR_D3D11_enable                           | 338 |
| 12.13. XR_KHR_D3D12_enable                           | 345 |
| 12.14. XR_KHR_loader_init                            | 352 |
| 12.15. XR_KHR_loader_init_android                    | 355 |
| 12.16. XR_KHR_opengl_enable                          | 357 |
| 12.17. XR_KHR_opengl_es_enable                       | 368 |
| 12.18. XR_KHR_swapchain_usage_input_attachment_bit   | 375 |
| 12.19. XR_KHR_visibility_mask                        | 376 |
| 12.20. XR_KHR_vulkan_enable                          | 382 |
| 12.21. XR_KHR_vulkan_enable2                         | 395 |
| 12.22. XR_KHR_vulkan_swapchain_format_list           | 413 |
| 12.23. XR_KHR_win32_convert_performance_counter_time | 416 |
| 12.24. XR_EXT_active_action_set_priority             | 419 |
| 12.25. XR_EXT_conformance_automation                 | 422 |
| 12.26. XR_EXT_debug_utils                            | 431 |
| 12.27. XR_EXT_dpad_binding                           | 457 |
| 12.28. XR_EXT_eye_gaze_interaction                   | 466 |

|   |     |
|---|-----|
| 12.29. XR_EXT_future                          | 473 |
| 12.30. XR_EXT_hand_interaction                | 493 |
| 12.31. XR_EXT_hand_joints_motion_range        | 506 |
| 12.32. XR_EXT_hand_tracking                   | 508 |
| 12.33. XR_EXT_hand_tracking_data_source       | 528 |
| 12.34. XR_EXT_performance_settings            | 533 |
| 12.35. XR_EXT_plane_detection                 | 544 |
| 12.36. XR_EXT_thermal_query                   | 571 |
| 12.37. XR_EXT_user_presence                   | 575 |
| 12.38. XR_EXT_view_configuration_depth_range  | 581 |
| 12.39. XR_EXT_win32_appcontainer_compatible   | 584 |
| 12.40. XR_ALMALENCE_digital_lens_control      | 585 |
| 12.41. XR_EPIC_view_configuration_fov         | 589 |
| 12.42. XR_FB_android_surface_swapchain_create | 591 |
| 12.43. XR_FB_body_tracking                    | 594 |
| 12.44. XR_FB_color_space                      | 612 |
| 12.45. XR_FB_composition_layer_alpha_blend    | 619 |
| 12.46. XR_FB_composition_layer_depth_test     | 622 |
| 12.47. XR_FB_composition_layer_image_layout   | 624 |
| 12.48. XR_FB_composition_layer_secure_content | 627 |
| 12.49. XR_FB_composition_layer_settings       | 630 |
| 12.50. XR_FB_display_refresh_rate             | 633 |
| 12.51. XR_FB_eye_tracking_social              | 640 |
| 12.52. XR_FB_face_tracking                    | 652 |
| 12.53. XR_FB_face_tracking2                   | 666 |
| 12.54. XR_FB_foveation                        | 706 |
| 12.55. XR_FB_foveation_configuration          | 714 |
| 12.56. XR_FB_foveation_vulkan                 | 717 |
| 12.57. XR_FB_hand_tracking_aim                | 719 |
| 12.58. XR_FB_hand_tracking_capsules           | 723 |
| 12.59. XR_FB_hand_tracking_mesh               | 726 |
| 12.60. XR_FB_haptic_amplitude_envelope        | 733 |
| 12.61. XR_FB_haptic_pcm                       | 735 |
| 12.62. XR_FB_keyboard_tracking                | 741 |
| 12.63. XR_FB_passthrough                      | 749 |
| 12.64. XR_FB_passthrough_keyboard_hands       | 779 |
| 12.65. XR_FB_render_model                     | 782 |
| 12.66. XR_FB_scene                            | 795 |

|   |      |
|---|------|
| 12.67. XR_FB_scene_capture .....                          | 811  |
| 12.68. XR_FB_space_warp .....                             | 816  |
| 12.69. XR_FB_spatial_entity .....                         | 820  |
| 12.70. XR_FB_spatial_entity_container .....               | 836  |
| 12.71. XR_FB_spatial_entity_query .....                   | 840  |
| 12.72. XR_FB_spatial_entity_sharing .....                 | 854  |
| 12.73. XR_FB_spatial_entity_storage .....                 | 860  |
| 12.74. XR_FB_spatial_entity_storage_batch .....           | 868  |
| 12.75. XR_FB_spatial_entity_user .....                    | 874  |
| 12.76. XR_FB_swapchain_update_state .....                 | 879  |
| 12.77. XR_FB_swapchain_update_state_android_surface ..... | 884  |
| 12.78. XR_FB_swapchain_update_state_opengl_es .....       | 886  |
| 12.79. XR_FB_swapchain_update_state_vulkan .....          | 889  |
| 12.80. XR_FB_touch_controller_pro .....                   | 892  |
| 12.81. XR_FB_touch_controller_proximity .....             | 896  |
| 12.82. XR_FB_triangle_mesh .....                          | 899  |
| 12.83. XR_HTC_anchor .....                                | 915  |
| 12.84. XR_HTC_facial_tracking .....                       | 921  |
| 12.85. XR_HTC_foveation .....                             | 947  |
| 12.86. XR_HTC_hand_interaction .....                      | 956  |
| 12.87. XR_HTC_passthrough .....                           | 958  |
| 12.88. XR_HTC_vive_wrist_tracker_interaction .....        | 967  |
| 12.89. XR_HUAWEI_controller_interaction .....             | 969  |
| 12.90. XR_META_automatic_layer_filter .....               | 972  |
| 12.91. XR_META_environment_depth .....                    | 973  |
| 12.92. XR_META_foveation_eye_tracked .....                | 997  |
| 12.93. XR_META_headset_id .....                           | 1004 |
| 12.94. XR_META_local_dimming .....                        | 1007 |
| 12.95. XR_META_passthrough_color_lut .....                | 1009 |
| 12.96. XR_META_passthrough_preferences .....              | 1021 |
| 12.97. XR_META_performance_metrics .....                  | 1025 |
| 12.98. XR_META_recommended_layer_resolution .....         | 1034 |
| 12.99. XR_META_spatial_entity_mesh .....                  | 1039 |
| 12.100. XR_META_touch_controller_plus .....               | 1044 |
| 12.101. XR_META_virtual_keyboard .....                    | 1047 |
| 12.102. XR_META_vulkan_swapchain_create_info .....        | 1088 |
| 12.103. XR_ML_compat .....                                | 1090 |
| 12.104. XR_ML_frame_end_info .....                        | 1093 |

|   |      |
|---|------|
| 12.105. XR_ML_global_dimmer                       | 1096 |
| 12.106. XR_ML_localization_map                    | 1098 |
| 12.107. XR_ML_marker_understanding                | 1119 |
| 12.108. XR_ML_user_calibration                    | 1157 |
| 12.109. XR_MND_headless                           | 1162 |
| 12.110. XR_MSFT_composition_layer_reprojection    | 1164 |
| 12.111. XR_MSFT_controller_model                  | 1170 |
| 12.112. XR_MSFT_first_person_observer             | 1183 |
| 12.113. XR_MSFT_hand_interaction                  | 1185 |
| 12.114. XR_MSFT_hand_tracking_mesh                | 1188 |
| 12.115. XR_MSFT_holographic_window_attachment     | 1208 |
| 12.116. XR_MSFT_perception_anchor_interop         | 1212 |
| 12.117. XR_MSFT_scene_marker                      | 1216 |
| 12.118. XR_MSFT_scene_understanding               | 1231 |
| 12.119. XR_MSFT_scene_understanding_serialization | 1284 |
| 12.120. XR_MSFT_secondary_view_configuration      | 1291 |
| 12.121. XR_MSFT_spatial_anchor                    | 1303 |
| 12.122. XR_MSFT_spatial_anchor_persistence        | 1310 |
| 12.123. XR_MSFT_spatial_graph_bridge              | 1323 |
| 12.124. XR_MSFT_unbounded_reference_space         | 1335 |
| 12.125. XR_OCULUS_audio_device_guid               | 1337 |
| 12.126. XR_OCULUS_external_camera                 | 1340 |
| 12.127. XR_OPPO_controller_interaction            | 1346 |
| 12.128. XR_QCOM_tracking_optimization_settings    | 1349 |
| 12.129. XR_ULTRALEAP_hand_tracking_forearm        | 1353 |
| 12.130. XR_VALVE_analog_threshold                 | 1356 |
| 12.131. XR_VARJO_composition_layer_depth_test     | 1359 |
| 12.132. XR_VARJO_environment_depth_estimation     | 1363 |
| 12.133. XR_VARJO_foveated_rendering               | 1365 |
| 12.134. XR_VARJO_marker_tracking                  | 1373 |
| 12.135. XR_VARJO_view_offset                      | 1385 |
| 12.136. XR_VARJO_xr4_controller_interaction       | 1388 |
| 12.137. XR_YVR_controller_interaction             | 1390 |
| 13. List of Provisional Extensions                | 1394 |
| 13.1. XR_EXTX_overlay                             | 1395 |
| 13.2. XR_HTCX_vive_tracker_interaction            | 1402 |
| 13.3. XR_MNDX_egl_enable                          | 1411 |
| 13.4. XR_MNDX_force_feedback_curl                 | 1413 |

|  |      |
|--|------|
| 14. List of Deprecated Extensions .....                          | 1421 |
| 14.1. XR_KHR_locate_spaces .....                                 | 1422 |
| 14.2. XR_KHR_maintenance1 .....                                  | 1432 |
| 14.3. XR_EXT_hp_mixed_reality_controller .....                   | 1435 |
| 14.4. XR_EXT_local_floor .....                                   | 1437 |
| 14.5. XR_EXT_palm_pose .....                                     | 1441 |
| 14.6. XR_EXT_samsung_odyssey_controller .....                    | 1444 |
| 14.7. XR_EXT_uuid .....  | 1446 |
| 14.8. XR_BD_controller_interaction .....                         | 1448 |
| 14.9. XR_HTC_vive_cosmos_controller_interaction .....            | 1453 |
| 14.10. XR_HTC_vive_focus3_controller_interaction .....           | 1456 |
| 14.11. XR_ML_ml2_controller_interaction .....                    | 1459 |
| 14.12. XR_MND_swapchain_usage_input_attachment_bit .....         | 1461 |
| 14.13. XR_OCULUS_android_session_state_enable .....              | 1463 |
| 14.14. XR_VARJO_quad_views .....                                 | 1465 |
| 15. Core Revisions (Informative) .....                           | 1468 |
| 15.1. Version 1.1 .....  | 1468 |
| 15.2. Loader Runtime and API Layer Negotiation Version 1.0 ..... | 1470 |
| 15.3. Version 1.0 .....  | 1471 |
| Appendix .....   | 1476 |
| Code Style Conventions .....                                     | 1476 |
| Application Binary Interface .....                               | 1476 |
| Android Notes .....  | 1497 |
| Glossary .....   | 1497 |
| Abbreviations .....  | 1499 |
| Dedication (Informative) .....                                   | 1501 |
| Contributors (Informative) .....                                 | 1503 |
| Index .....  | 1507 |



# Preamble

Copyright (c) 2017-2024, The Khronos Group Inc.

This Specification is protected by copyright laws and contains material proprietary to Khronos. Except as described by these terms, it or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos.

Khronos grants a conditional copyright license to use and reproduce the unmodified Specification for any purpose, without fee or royalty, EXCEPT no licenses to any patent, trademark or other intellectual property rights are granted under these terms.

Khronos makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this Specification, including, without limitation: merchantability, fitness for a particular purpose, non-infringement of any intellectual property, correctness, accuracy, completeness, timeliness, and reliability. Under no circumstances will Khronos, or any of its Promoters, Contributors or Members, or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

This document contains extensions which are not ratified by Khronos, and as such is not a ratified Specification, though it contains text from (and is a superset of) the ratified OpenXR Specification that can be found at <https://registry.khronos.org/OpenXR/specs/1.1-khr/html/xrspec.html> (core with KHR extensions).

The Khronos Intellectual Property Rights Policy defines the terms 'Scope', 'Compliant Portion', and 'Necessary Patent Claims'.

Some parts of this Specification are purely informative and so are EXCLUDED from the Scope of this Specification. The [Document Conventions](#) section of the [Introduction](#) defines how these parts of the Specification are identified.

Where this Specification uses technical terminology, defined in the [Glossary](#) or otherwise, that refer to enabling technologies that are not expressly set forth in this Specification, those enabling technologies are EXCLUDED from the Scope of this Specification. For clarity, enabling technologies not disclosed with particularity in this Specification (e.g. semiconductor manufacturing technology, hardware architecture, processor architecture or microarchitecture, memory architecture, compiler technology, object oriented technology, basic operating system technology, compression technology, algorithms, and so on) are NOT to be considered expressly set forth; only those application program interfaces and data structures disclosed with particularity are included in the Scope of this Specification.

For purposes of the Khronos Intellectual Property Rights Policy as it relates to the definition of Necessary Patent Claims, all recommended or optional features, behaviors and functionality set forth in this Specification, if implemented, are considered to be included as Compliant Portions.

Khronos® and Vulkan® are registered trademarks, and glTF™ is a trademark of The Khronos Group Inc. OpenXR™ is a trademark owned by The Khronos Group Inc. and is registered as a trademark in China, the European Union, Japan and the United Kingdom. OpenGL® is a registered trademark and the OpenGL ES™ and OpenGL SC™ logos are trademarks of Hewlett Packard Enterprise used under license by Khronos. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

# Chapter 1. Introduction

This chapter is informative except for the section on [Normative Terminology](#).

This document, referred to as the "OpenXR Specification" or just the "Specification" hereafter, describes OpenXR: what it is, how it acts, and what is required to implement it. We assume that the reader has a basic understanding of computer graphics and the technologies involved in virtual and augmented reality. This means familiarity with the essentials of computer graphics algorithms and terminology, modern GPUs (Graphic Processing Units), tracking technologies, head mounted devices, and input modalities.

The canonical version of the Specification is available in the official OpenXR Registry, located at URL

<https://registry.khronos.org/OpenXR>

## 1.1. What is OpenXR?

OpenXR is an API (Application Programming Interface) for XR applications. XR refers to a continuum of real-and-virtual combined environments generated by computers through human-machine interaction and is inclusive of the technologies associated with virtual reality (VR), augmented reality (AR) and mixed reality (MR). OpenXR is the interface between an application and an in-process or out-of-process "XR runtime system", or just "runtime" hereafter. The runtime may handle such functionality as frame composition, peripheral management, and raw tracking information.

Optionally, a runtime may support device layer plugins which allow access to a variety of hardware across a commonly defined interface.

## 1.2. The Programmer's View of OpenXR

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

A typical OpenXR program begins with a call to create an instance which establishes a connection to a runtime. Then a call is made to create a system which selects for use a physical display and a subset of input, tracking, and graphics devices. Subsequently a call is made to create buffers into which the application will render one or more views using the appropriate graphics APIs for the platform. Finally calls are made to create a session and begin the application's XR rendering loop.

## 1.3. The Implementor's View of OpenXR

To the runtime implementor, OpenXR is a set of functions that control the operation of the XR system and establishes the lifecycle of a XR application.

The implementor’s task is to provide a software library on the host which implements the OpenXR API, while mapping the work for each OpenXR function to the graphics hardware as appropriate for the capabilities of the device.

## 1.4. Our View of OpenXR

We view OpenXR as a mechanism for interacting with VR/AR/MR systems in a platform-agnostic way.

We expect this model to result in a specification that satisfies the needs of both programmers and runtime implementors. It does not, however, necessarily provide a model for implementation. A runtime implementation **must** produce results conforming to those produced by the specified methods, but **may** carry out particular procedures in ways that are more efficient than the one specified.

## 1.5. Filing Bug Reports

Issues with and bug reports on the OpenXR Specification and the API Registry **can** be filed in the Khronos OpenXR GitHub repository, located at URL

<https://github.com/KhronosGroup/OpenXR-Docs>

Please tag issues with appropriate labels, such as “Specification”, “Ref Pages” or “Registry”, to help us triage and assign them appropriately. Unfortunately, GitHub does not currently let users who do not have write access to the repository set GitHub labels on issues. In the meantime, they **can** be added to the title line of the issue set in brackets, e.g. “[Specification]”.

## 1.6. Document Conventions

The OpenXR specification is intended for use by both implementors of the API and application developers seeking to make use of the API, forming a contract between these parties. Specification text may address either party; typically the intended audience can be inferred from context, though some sections are defined to address only one of these parties. (For example, Valid Usage sections only address application developers). Any requirements, prohibitions, recommendations or options defined by normative terminology are imposed only on the audience of that text.

### 1.6.1. Normative Terminology

The key words **must**, **required**, **should**, **may**, and **optional** in this document, when denoted as above, are to be interpreted as described in RFC 2119:

<https://tools.ietf.org/html/rfc2119>

#### **must**

When used alone, this word, or the term **required**, means that the definition is an absolute requirement of the specification. When followed by **not** (“**must not**”), the phrase means that the

definition is an absolute prohibition of the specification.

### **should**

When used alone, this word means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course. When followed by **not** (“**should not**”), the phrase means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications **should** be understood and the case carefully weighed before implementing any behavior described with this label.

### **may**

This word, or the adjective **optional**, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

The additional terms **can** and **cannot** are to be interpreted as follows:

### **can**

This word means that the particular behavior described is a valid choice for an application, and is never used to refer to runtime behavior.

### **cannot**

This word means that the particular behavior described is not achievable by an application, for example, an entry point does not exist.



There is an important distinction between **cannot** and **must not**, as used in this Specification. **Cannot** means something the application literally is unable to express or accomplish through the API, while **must not** means something that the application is capable of expressing through the API, but that the consequences of doing so are undefined and potentially unrecoverable for the runtime.

# Chapter 2. Fundamentals

## 2.1. API Version Numbers and Semantics

Multi-part version numbers are used in several places in the OpenXR API.

```
// Provided by XR_VERSION_1_0
typedef uint64_t XrVersion;
```

In each such use, the API major version number, minor version number, and patch version number are packed into a 64-bit integer, referred to as `XrVersion`, as follows:

### Version Numbers

- The major version number is a 16-bit integer packed into bits 63-48.
- The minor version number is a 16-bit integer packed into bits 47-32.
- The patch version number is a 32-bit integer packed into bits 31-0.

Differences in any of the version numbers indicate a change to the API, with each part of the version number indicating a different scope of change, as follows.



#### Note

The rules below apply to OpenXR versions 1.0 or later. Prerelease versions of OpenXR may use different rules for versioning.

A difference in patch version numbers indicates that some usually small part of the specification or header has been modified, typically to fix a bug, and **may** have an impact on the behavior of existing functionality. Differences in the patch version number **must** affect neither full compatibility nor backwards compatibility between two versions, nor **may** it add additional interfaces to the API. Runtimes **may** use patch version number to determine whether to enable implementation changes, such as bug fixes, that impact functionality. Runtimes **should** document any changes that are tied to the patch version. Application developers **should** retest their application on all runtimes they support after compiling with a new version.

A difference in minor version numbers indicates that some amount of new functionality has been added. This will usually include new interfaces in the header, and **may** also include behavior changes and bug fixes. Functionality **may** be deprecated in a minor revision, but **must** not be removed. When a new minor version is introduced, the patch version continues where the last minor version left off, making patch versions unique inside major versions. Differences in the minor version number **should**

not affect backwards compatibility, but will affect full compatibility.

A difference in major version numbers indicates a large set of changes to the API, potentially including new functionality and header interfaces, behavioral changes, removal of deprecated features, modification or outright replacement of any feature, and is thus very likely to break compatibility. Differences in the major version number will typically require significant modification to application code in order for it to function properly.

The following table attempts to detail the changes that **may** occur versus when they **must** not be updated during an update to any of the major, minor, or patch version numbers:

Table 1. Scenarios Which May Cause a Version Change

| <i>Reason</i>                           | <i>Major Version</i> | <i>Minor Version</i> | <i>Patch Version</i> |
|---|----------------------|----------------------|----------------------|
| <i>Extensions Added/Removed*</i>        | <b>may</b>           | <b>may</b>           | <b>may</b>           |
| <i>Spec-Optional Behavior Changed*</i>  | <b>may</b>           | <b>may</b>           | <b>may</b>           |
| <i>Spec Required Behavior Changed*</i>  | <b>may</b>           | <b>may</b>           | <b>must not</b>      |
| <i>Core Interfaces Added*</i>           | <b>may</b>           | <b>may</b>           | <b>must not</b>      |
| <i>Weak Deprecation*</i>                | <b>may</b>           | <b>may</b>           | <b>must not</b>      |
| <i>Strong Deprecation*</i>              | <b>may</b>           | <b>must not</b>      | <b>must not</b>      |
| <i>Core Interfaces Changed/Removed*</i> | <b>may</b>           | <b>must not</b>      | <b>must not</b>      |

In the above table, the following identify the various cases in detail:

|  |  |
|--|--|
| <i>Extensions Added/Removed</i>                | An extension <b>may</b> be added or removed with a change at this patch level.   |
| <i>Specification-Optional Behavior Changed</i> | Some <b>optional</b> behavior laid out in this specification has changed. Usually this will involve a change in behavior that is marked with the normative language <b>should</b> or <b>may</b> . For example, a runtime that previously did not validate a particular use case <b>may</b> now begin validating that use case. |
| <i>Specification-Required Behavior Changed</i> | A behavior of runtimes that is required by this specification <b>may</b> have changed. For example, a previously <b>optional</b> validation <b>may</b> now have become mandatory for runtimes.   |
| <i>Core Interfaces Added</i>                   | New interfaces <b>may</b> have been added to this specification (and to the OpenXR header file) in revisions at this level.  |

### *Weak Deprecation*

An interface **may** have been weakly deprecated at this level. This **may** happen if there is now a better way to accomplish the same thing. Applications making this call **should** behave the same as before the deprecation, but following the new path **may** be more performant, lower latency, or otherwise yield better results. It is possible that some runtimes **may** choose to give run-time warnings that the feature has been weakly deprecated and will likely be strongly deprecated or removed in the future.

### *Strong Deprecation*

An interface **may** have been strongly deprecated at this level. This means that the interface **must** still exist (so applications that are compiled against it will still run) but it **may** now be a no-op, or it **may** be that its behavior has been significantly changed. It **may** be that this functionality is no longer necessary, or that its functionality has been subsumed by another call. This **should** not break an application, but some behavior **may** be different or unanticipated.

### *Interfaces Changed/Removed*

An interface **may** have been changed — with different parameters or return types — at this level. An interface or feature **may** also have been removed entirely. It is almost certain that rebuilding applications will be required.

---

## 2.2. String Encoding

This API uses strings as input and output for some functions. Unless otherwise specified, all such strings are **NULL** terminated UTF-8 encoded case-sensitive character arrays.

## 2.3. Threading Behavior

The OpenXR API is intended to provide scalable performance when used on multiple host threads. All functions **must** support being called concurrently from multiple threads, but certain parameters, or components of parameters are defined to be externally synchronized. This means that the caller **must** guarantee that no more than one thread is using such a parameter at a given time.

More precisely, functions use simple stores to update software structures representing objects. A parameter declared as externally synchronized **may** have its software structures updated at any time during the host execution of the function. If two functions operate on the same object and at least one of the functions declares the object to be externally synchronized, then the caller **must** guarantee not only that the functions do not execute simultaneously, but also that the two functions are separated by an appropriate memory barrier if needed.

For all functions which destroy an object handle, the application **must** externally synchronize the object handle parameter and any child handles.



## Externally Synchronized Parameters

- The **instance** parameter, and any child handles, in [xrDestroyInstance](#)
- The **session** parameter, and any child handles, in [xrDestroySession](#)
- The **space** parameter, and any child handles, in [xrDestroySpace](#)
- The **swapchain** parameter, and any child handles, in [xrDestroySwapchain](#)
- The **actionSet** parameter, and any child handles, in [xrDestroyActionSet](#)
- The **action** parameter, and any child handles, in [xrDestroyAction](#)
- The **objectHandle** member of the **nameInfo** parameter in [xrSetDebugUtilsObjectNameEXT](#)
- The **instance** parameter, and any child handles, in [xrCreateDebugUtilsMessengerEXT](#)
- The **messenger** parameter in [xrDestroyDebugUtilsMessengerEXT](#)
- The **anchor** parameter, and any child handles, in [xrDestroySpatialAnchorMSFT](#)
- The **nodeBinding** parameter, and any child handles, in [xrDestroySpatialGraphNodeBindingMSFT](#)
- The **handTracker** parameter, and any child handles, in [xrDestroyHandTrackerEXT](#)
- The **bodyTracker** parameter, and any child handles, in [xrDestroyBodyTrackerFB](#)
- The **sceneObserver** parameter, and any child handles, in [xrDestroySceneObserverMSFT](#)
- The **scene** parameter, and any child handles, in [xrDestroySceneMSFT](#)
- The **facialTracker** parameter, and any child handles, in [xrDestroyFacialTrackerHTC](#)
- The **profile** parameter, and any child handles, in [xrDestroyFoveationProfileFB](#)
- The **mesh** parameter, and any child handles, in [xrDestroyTriangleMeshFB](#)
- The **passthrough** parameter, and any child handles, in [xrDestroyPassthroughFB](#)
- The **layer** parameter, and any child handles, in [xrDestroyPassthroughLayerFB](#)
- The **instance** parameter, and any child handles, in [xrDestroyGeometryInstanceFB](#)
- The **markerDetector** parameter, and any child handles, in [xrDestroyMarkerDetectorML](#)
- The **map** parameter, and any child handles, in [xrDestroyExportedLocalizationMapML](#)
- The **spatialAnchorStore** parameter, and any child handles, in [xrDestroySpatialAnchorStoreConnectionMSFT](#)
- The **faceTracker** parameter, and any child handles, in [xrDestroyFaceTrackerFB](#)
- The **eyeTracker** parameter, and any child handles, in [xrDestroyEyeTrackerFB](#)
- The **keyboard** parameter, and any child handles, in [xrDestroyVirtualKeyboardMETA](#)
- The **user** parameter, and any child handles, in [xrDestroySpaceUserFB](#)
- The **colorLut** parameter, and any child handles, in [xrDestroyPassthroughColorLutMETA](#)

- The `faceTracker` parameter, and any child handles, in `xrDestroyFaceTracker2FB`
- The `environmentDepthProvider` parameter, and any child handles, in `xrDestroyEnvironmentDepthProviderMETA`
- The `swapchain` parameter, and any child handles, in `xrDestroyEnvironmentDepthSwapchainMETA`
- The `passthrough` parameter, and any child handles, in `xrDestroyPassthroughHTC`
- The `planeDetector` parameter, and any child handles, in `xrDestroyPlaneDetectorEXT`
- The `future` member of the `cancelInfo` parameter in `xrCancelFutureEXT`

### Implicit Externally Synchronized Parameters

- The `session` parameter by any other `xrWaitFrame` call in `xrWaitFrame`
- The `session` parameter by any other `xrBeginFrame` or `xrEndFrame` call in `xrBeginFrame`
- The `session` parameter by any other `xrBeginFrame` or `xrEndFrame` call in `xrEndFrame`
- The `XrInstance` used to create `messenger`, and all of its child handles in `xrDestroyDebugUtilsMessengerEXT`
- The buffers returned from calls to `xrTriangleMeshGetVertexBufferFB` and `xrTriangleMeshGetIndexBufferFB` on `mesh` in `xrDestroyTriangleMeshFB`

## 2.4. Multiprocessing Behavior

The OpenXR API does not explicitly recognize nor require support for multiple processes using the runtime simultaneously, nor does it prevent a runtime from providing such support.

## 2.5. Runtime

An OpenXR runtime is software which implements the OpenXR API. There **may** be more than one OpenXR runtime installed on a system, but only one runtime can be active at any given time.

## 2.6. Extensions

OpenXR is an extensible API that grows through the addition of new features. Similar to other Khronos APIs, extensions **may** expose new OpenXR functions or modify the behavior of existing OpenXR functions. Extensions are **optional**, and therefore **must** be enabled by the application before the extended functionality is made available. Because extensions are **optional**, they **may** be implemented only on a subset of runtimes, graphics platforms, or operating systems. Therefore, an application **should** first query which extensions are available before enabling.

The application queries the available list of extensions using the [xrEnumerateInstanceExtensionProperties](#) function. Once an application determines which extensions are supported, it **can** enable some subset of them during the call to [xrCreateInstance](#).

OpenXR extensions have unique names that convey information about what functionality is provided. The names have the following format:

### Extension Name Formatting

- The prefix "XR\_" to identify this as an OpenXR extension
- A string identifier for the vendor tag, which corresponds to the company or group exposing the extension. The vendor tag **must** use only uppercase letters and decimal digits. Some examples include:
  - "KHR" for Khronos extensions, supported by multiple vendors.
  - "EXT" for non-Khronos extensions supported by multiple vendors.
- An underscore "\_".
- A string uniquely identifying the extension. The string is a compound of substrings which **must** use only lower case letters and decimal digits. The substrings are delimited with single underscores.

For example: [XR\\_KHR\\_composition\\_layer\\_cube](#) is an OpenXR extension created by the Khronos (KHR) OpenXR Working Group to support cube composition layers.

The public list of available extensions known and configured for inclusion in this document at the time of this specification being generated appears in the [List of Extensions](#) appendix at the end of this document.

## 2.7. API Layers

OpenXR is designed to be a layered API, which means that a user or application **may** insert API layers between the application and the runtime implementation. These API layers provide additional functionality by intercepting OpenXR functions from the layer above and performing different operations than would otherwise be performed without the layer. In the simplest cases, the layer simply calls the next layer down with the same arguments, but a more complex layer **may** implement API functionality that is not present in the layers or runtime below it. This mechanism is essentially an architected "function shimming" or "intercept" feature that is designed into OpenXR and meant to replace more informal methods of "hooking" API calls.

### 2.7.1. Examples of API Layers

## Validation Layer

The layered API approach employed by OpenXR allows for potentially expensive validation of correct API usage to be implemented in a "validation" layer. Such a layer allows the application developer to develop their application with a validation layer active to ensure that the application is using the API correctly. A validation layer confirms that the application has set up object state correctly, has provided the required data for each function, ensures that required resources are available, etc. If a validation layer detects a problem, it issues an error message that **can** be logged or captured by the application via a callback. After the developer has determined that the application is correct, they turn off a validation layer to allow the application to run in a production environment without repeatedly incurring the validation expense. (Note that some validation of correct API usage is required to be implemented by the runtime.)

## API Logging Layer

Another example of an API layer is an API logging layer that simply serializes all the API calls to an output sink in a text format, including printing out argument values and structure contents.

## API Trace Layer

A related API trace layer produces a trace file that contains all the information provided to the API so that the trace file can be played back by a replay program.

### 2.7.2. Naming API Layers

To organize API layer names and prevent collisions in the API layer name namespace, API layers **must** be named using the following convention:

```
XR_API_LAYER_<VENDOR-TAG>_short_name
```

Vendors are responsible for registering a vendor tag with the OpenXR working group, and just like for implementors, they must maintain their vendor namespace.

Example of an API layer name produced by the Acme company for the "check best practices" API layer:

```
XR_API_LAYER_ACME_check_best_practices
```

### 2.7.3. Activating API Layers

#### Application Activation

Applications **can** determine the API layers that are available to them by calling the [xrEnumerateApiLayerProperties](#) function to obtain a list of available API layers. Applications then **can** select the desired API layers from this list and provide them to the [xrCreateInstance](#) function when

creating an instance.

## System Activation

Application users or users performing roles such as system integrator or system administrator **may** configure a system to activate API layers without involvement from the applications. These platform-dependent steps **may** include the installation of API layer-related files, setting environment variables, or other platform-specific operations. The options that are available for configuring the API layers in this manner are also dependent on the platform and/or runtime.

### 2.7.4. API Layer Extensions

API layers **may** implement OpenXR functions that are not supported by the underlying runtime. In order to expose these new features, the API layer **must** expose this functionality in the form of an OpenXR [extension](#). It **must** not expose new OpenXR functions without an associated extension.

For example, an OpenXR API-logging API layer might expose an API function to allow the application to turn logging on for only a portion of its execution. Since new functions **must** be exposed through an extension, the vendor has created an extension called `XR_ACME_logging_on_off` to contain these new functions. The application **should** query if the API layer supports the extension and then, only if it exists, enable both the extension and the API layer by name during `xrCreateInstance`.

To find out what extensions an API layer supports, an application **must** first verify that the API layer exists on the current system by calling `xrEnumerateApiLayerProperties`. After verifying an API layer of interest exists, the application then **should** call `xrEnumerateInstanceExtensionProperties` and provide the API layer name as the first parameter. This will return the list of extensions implemented by that API layer.

## 2.8. Type Aliasing

Type aliasing refers to the situation in which the actual type of a element does not match the declared type. Some C and C++ compilers assume that the actual type matches the declared type in some configurations, and **may** be so configured by default at common optimization levels. In such a compiler configured with that assumption, violating the assumption **may** produce undefined behavior. This compiler feature is typically referred to as "strict aliasing," and it **can** usually be enabled or disabled via compiler options. The OpenXR specification **does not** support strict aliasing, as there are some cases in which an application intentionally provides a struct with a type that differs from the declared type. For example, `XrFrameEndInfo::layers` is an array of type `const XrCompositionLayerBaseHeader` code: `* const`. However, each element of the array **must** be of one of the specific layer types, such as `XrCompositionLayerQuad`. Similarly, `xrEnumerateSwapchainImages` accepts an array of `XrSwapchainImageBaseHeader`, whereas the actual type passed **must** be an array of a type such as `XrSwapchainImageVulkanKHR`.

For OpenXR to work correctly, the compiler **must** support the type aliasing described here.

```
// Provided by XR_VERSION_1_0
#if !defined(XR_MAY_ALIAS)
#if defined(__clang__) || (defined(__GNUC__) && (__GNUC__ > 4))
#define XR_MAY_ALIAS __attribute__((__may_alias__))
#else
#define XR_MAY_ALIAS
#endif
#endif
```

As a convenience, some types and pointers that are known at specification time to alias values of different types have been annotated with the [XR\\_MAY\\_ALIAS](#) definition. If this macro is not defined before including OpenXR headers, and a new enough Clang or GCC compiler is used, it is defined to a compiler-specific attribute annotation to inform these compilers that those pointers **may** alias. However, there is no guarantee that all aliasing types or pointers have been correctly marked with this macro, so thorough testing is still recommended if you choose (at your own risk) to permit your compiler to perform type-based aliasing analysis.

## 2.9. Valid Usage

Valid usage defines a set of conditions which **must** be met in order to achieve well-defined run-time behavior in an application. These conditions depend only on API state, and the parameters or objects whose usage is constrained by the condition.

Some valid usage conditions have dependencies on runtime limits or feature availability. It is possible to validate these conditions against the API's minimum or maximum supported values for these limits and features, or some subset of other known values.

Valid usage conditions **should** apply to a function or structure where complete information about the condition would be known during execution of an application. This is such that a validation API layer or linter **can** be written directly against these statements at the point they are specified.

### 2.9.1. Implicit Valid Usage

Some valid usage conditions apply to all functions and structures in the API, unless explicitly denoted otherwise for a specific function or structure. These conditions are considered implicit. Implicit valid usage conditions are described in detail below.

### 2.9.2. Valid Usage for Object Handles

Any input XR parameter to a function that is an object handle **must** be a valid object handle, unless otherwise specified. An object handle is valid if and only if all of the following conditions hold:

## Object Handle Validity Conditions

- It has been created or allocated by a previous, successful call to the API.
- It has not been destroyed by a previous call to the API.
- Its parent handle is also valid.

There are contexts in which an object handle is **optional** or otherwise unspecified. In those cases, the API uses `XR_NULL_HANDLE`, which has the integer value `0`.

### 2.9.3. Valid Usage for Pointers

Any parameter that is a pointer **must** be a valid pointer when the specification indicates that the runtime uses the pointer. A pointer is valid if and only if it points at memory containing values of the number and type(s) expected by the function, and all fundamental types accessed through the pointer (e.g. as elements of an array or as members of a structure) satisfy the alignment requirements of the host processor.

### 2.9.4. Valid Usage for Enumerated Types

Any parameter of an enumerated type **must** be a valid enumerant for that type. An enumerant is valid if and only if the enumerant is defined as part of the enumerated type in question.

### 2.9.5. Valid Usage for Flags

A collection of flags is represented by a bitmask using the type `XrFlags64`:

```
typedef uint64_t XrFlags64;
```

Bitmasks are passed to many functions and structures to compactly represent options and are stored in memory defined by the `XrFlags64` type. But the API does not use the `XrFlags64` type directly. Instead, a `Xr*Flags` type is used which is an alias of the `XrFlags64` type. The API also defines a set of constant bit definitions used to set the bitmasks.

Any `Xr*Flags` member or parameter used in the API **must** be a valid combination of bit flags. A valid combination is either zero or the bitwise **OR** of valid bit flags. A bit flag is valid if and only if:

## Bit Flag Validity

- The bit flag is one of the constant bit definitions defined by the same `Xr*Flags` type as the `Xr*Flags` member or parameter. (Valid flag values **may** also be defined by extensions but will appear in the specification with all other valid flag values for that type.)
- The flag is allowed in the context in which it is being used. For example, in some cases, certain bit flags or combinations of bit flags are mutually exclusive.

### 2.9.6. Valid Usage for Structure Types

Any parameter that is a structure containing a `type` member **must** have a value of `type` which is a valid `XrStructureType` value matching the type of the structure. As a general rule, the name of this value is obtained by taking the structure name, stripping the leading `Xr`, prefixing each capital letter with an underscore, converting the entire resulting string to upper case, and prefixing it with `XR_TYPE_`.

The only exceptions to this rule are API and Operating System names which are converted in a way that produces a more readable value:

#### Structure Type Format Exceptions

- OpenGL ⇒ `_OPENGL`
- OpenGL ES ⇒ `_OPENGL_ES`
- EGL ⇒ `_EGL`
- D3D ⇒ `_D3D`

### 2.9.7. Valid Usage for Structure Pointer Chains

Any structure containing a `void* next` member **must** have a value of `next` that is either `NULL`, or points to a valid structure that also contains `type` and `next` member values. The set of structures connected by `next` pointers is referred to as a `next` chain.

In order to use a structure type defined by an extension in a `next` chain, the proper extension **must** have been previously enabled during `xrCreateInstance`. A runtime **must** ignore all unrecognized structures in a `next` chain, including those associated with an extension that has not been enabled.

Some structures for use in a chain are described in the core OpenXR specification and are mentioned in the Member Descriptions. Any structure described in this document intended for use in a chain is mentioned in a "See also" list in the implicit valid usage of the structure they chain to. Most chained structures are associated with extensions, and are described in the base OpenXR Specification under the [List of Extensions](#). Vendor-specific extensions **may** be found there as well, or **may** only be available from the vendor's website or internal document repositories.



Unless otherwise specified: Chained structs which are output structs **may** be modified by the runtime with the exception of the type and next fields. Upon return from any function, all type and next fields in the chain **must** be unmodified.

## Useful Base Structures

As a convenience to runtimes and layers needing to iterate through a structure pointer chain, the OpenXR API provides the following base structures:

The [XrBaseInStructure](#) structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrBaseInStructure {
    XrStructureType          type;
    const struct XrBaseInStructure* next;
} XrBaseInStructure;
```

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- **next** is `NULL` or a pointer to the next structure in a structure chain.

[XrBaseInStructure](#) **can** be used to facilitate iterating through a read-only structure pointer chain.

The [XrBaseOutStructure](#) structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrBaseOutStructure {
    XrStructureType          type;
    struct XrBaseOutStructure* next;
} XrBaseOutStructure;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure. This base structure itself has no associated `XrStructureType` value.
- `next` is `NULL` or a pointer to the next structure in a structure chain.

`XrBaseOutStructure` can be used to facilitate iterating through a structure pointer chain that returns data back to the application.

These structures allow for some type safety and can be used by OpenXR API functions that operate on generic inputs and outputs.

### Next Chain Structure Uniqueness

Applications **should** ensure that they create and insert no more than one occurrence of each type of extension structure in a given `next` chain. Other components of OpenXR (such as the OpenXR loader or an API Layer) **may** insert duplicate structures into this chain. This provides those components the ability to update a structure that appears in the `next` chain by making a modified copy of that same structure and placing the new version at the beginning of the chain. The benefit of allowing this duplication is each component is no longer required to create a copy of the entire `next` chain just to update one structure. When duplication is present, all other OpenXR components **must** process only the first instance of a structure of a given type, and then ignore all instances of a structure of that same type.

If a component makes such a structure copy, and the original structure is also used to return content, then that component **must** copy the necessary content from the copied structure and into the original version of the structure upon completion of the function prior to proceeding back up the call stack. This is to ensure that OpenXR behavior is consistent whether or not that particular OpenXR component is present and/or enabled on the system.

### 2.9.8. Valid Usage for Nested Structures

The above conditions also apply recursively to members of structures provided as input to a function, either as a direct argument to the function, or themselves a member of another structure.

Specifics on valid usage of each function are covered in their individual sections.

## 2.10. Return Codes

The core API is designed to capture most, but not all, instances of incorrect usage. As such, most functions provide return codes. Functions in the API return their status via return codes that are in one of the two categories below.

## Return Code Categories

- Successful completion codes are returned when a function needs to communicate success or status information. All successful completion codes are non-negative values.
- Run time error codes are returned when a function needs to communicate a failure that could only be detected at run time. All run time error codes are negative values.

```
typedef enum XrResult {
    XR_SUCCESS = 0,
    XR_TIMEOUT_EXPIRED = 1,
    XR_SESSION_LOSS_PENDING = 3,
    XR_EVENT_UNAVAILABLE = 4,
    XR_SPACE_BOUNDS_UNAVAILABLE = 7,
    XR_SESSION_NOT_FOCUSED = 8,
    XR_FRAME_DISCARDED = 9,
    XR_ERROR_VALIDATION_FAILURE = -1,
    XR_ERROR_RUNTIME_FAILURE = -2,
    XR_ERROR_OUT_OF_MEMORY = -3,
    XR_ERROR_API_VERSION_UNSUPPORTED = -4,
    XR_ERROR_INITIALIZATION_FAILED = -6,
    XR_ERROR_FUNCTION_UNSUPPORTED = -7,
    XR_ERROR_FEATURE_UNSUPPORTED = -8,
    XR_ERROR_EXTENSION_NOT_PRESENT = -9,
    XR_ERROR_LIMIT_REACHED = -10,
    XR_ERROR_SIZE_INSUFFICIENT = -11,
    XR_ERROR_HANDLE_INVALID = -12,
    XR_ERROR_INSTANCE_LOST = -13,
    XR_ERROR_SESSION_RUNNING = -14,
    XR_ERROR_SESSION_NOT_RUNNING = -16,
    XR_ERROR_SESSION_LOST = -17,
    XR_ERROR_SYSTEM_INVALID = -18,
    XR_ERROR_PATH_INVALID = -19,
    XR_ERROR_PATH_COUNT_EXCEEDED = -20,
    XR_ERROR_PATH_FORMAT_INVALID = -21,
    XR_ERROR_PATH_UNSUPPORTED = -22,
    XR_ERROR_LAYER_INVALID = -23,
    XR_ERROR_LAYER_LIMIT_EXCEEDED = -24,
    XR_ERROR_SWAPCHAIN_RECT_INVALID = -25,
    XR_ERROR_SWAPCHAIN_FORMAT_UNSUPPORTED = -26,
    XR_ERROR_ACTION_TYPE_MISMATCH = -27,
    XR_ERROR_SESSION_NOT_READY = -28,
    XR_ERROR_SESSION_NOT_STOPPING = -29,
    XR_ERROR_TIME_INVALID = -30,
```

```

XR_ERROR_REFERENCE_SPACE_UNSUPPORTED = -31,
XR_ERROR_FILE_ACCESS_ERROR = -32,
XR_ERROR_FILE_CONTENTS_INVALID = -33,
XR_ERROR_FORM_FACTOR_UNSUPPORTED = -34,
XR_ERROR_FORM_FACTOR_UNAVAILABLE = -35,
XR_ERROR_API_LAYER_NOT_PRESENT = -36,
XR_ERROR_CALL_ORDER_INVALID = -37,
XR_ERROR_GRAPHICS_DEVICE_INVALID = -38,
XR_ERROR_POSE_INVALID = -39,
XR_ERROR_INDEX_OUT_OF_RANGE = -40,
XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED = -41,
XR_ERROR_ENVIRONMENT_BLEND_MODE_UNSUPPORTED = -42,
XR_ERROR_NAME_DUPLICATED = -44,
XR_ERROR_NAME_INVALID = -45,
XR_ERROR_ACTIONSET_NOT_ATTACHED = -46,
XR_ERROR_ACTIONSETS_ALREADY_ATTACHED = -47,
XR_ERROR_LOCALIZED_NAME_DUPLICATED = -48,
XR_ERROR_LOCALIZED_NAME_INVALID = -49,
XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING = -50,
XR_ERROR_RUNTIME_UNAVAILABLE = -51,
// Provided by XR_VERSION_1_1
XR_ERROR_EXTENSION_DEPENDENCY_NOT_ENABLED = -1000710001,
// Provided by XR_VERSION_1_1
XR_ERROR_PERMISSION_INSUFFICIENT = -1000710000,
// Provided by XR_KHR_android_thread_settings
XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR = -1000003000,
// Provided by XR_KHR_android_thread_settings
XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR = -1000003001,
// Provided by XR_MSFT_spatial_anchor
XR_ERROR_CREATE_SPATIAL_ANCHOR_FAILED_MSFT = -1000039001,
// Provided by XR_MSFT_secondary_view_configuration
XR_ERROR_SECONDARY_VIEW_CONFIGURATION_TYPE_NOT_ENABLED_MSFT = -1000053000,
// Provided by XR_MSFT_controller_model
XR_ERROR_CONTROLLER_MODEL_KEY_INVALID_MSFT = -1000055000,
// Provided by XR_MSFT_composition_layer_reprojection
XR_ERROR_REPROJECTION_MODE_UNSUPPORTED_MSFT = -1000066000,
// Provided by XR_MSFT_scene_understanding
XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT = -1000097000,
// Provided by XR_MSFT_scene_understanding
XR_ERROR_SCENE_COMPONENT_ID_INVALID_MSFT = -1000097001,
// Provided by XR_MSFT_scene_understanding
XR_ERROR_SCENE_COMPONENT_TYPE_MISMATCH_MSFT = -1000097002,
// Provided by XR_MSFT_scene_understanding
XR_ERROR_SCENE_MESH_BUFFER_ID_INVALID_MSFT = -1000097003,
// Provided by XR_MSFT_scene_understanding
XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT = -1000097004,
// Provided by XR_MSFT_scene_understanding
XR_ERROR_SCENE_COMPUTE_CONSISTENCY_MISMATCH_MSFT = -1000097005,

```

```

// Provided by XR_FB_display_refresh_rate
XR_ERROR_DISPLAY_REFRESH_RATE_UNSUPPORTED_FB = -1000101000,
// Provided by XR_FB_color_space
XR_ERROR_COLOR_SPACE_UNSUPPORTED_FB = -1000108000,
// Provided by XR_FB_spatial_entity
XR_ERROR_SPACE_COMPONENT_NOT_SUPPORTED_FB = -1000113000,
// Provided by XR_FB_spatial_entity
XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB = -1000113001,
// Provided by XR_FB_spatial_entity
XR_ERROR_SPACE_COMPONENT_STATUS_PENDING_FB = -1000113002,
// Provided by XR_FB_spatial_entity
XR_ERROR_SPACE_COMPONENT_STATUS_ALREADY_SET_FB = -1000113003,
// Provided by XR_FB_passthrough
XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB = -1000118000,
// Provided by XR_FB_passthrough
XR_ERROR_FEATURE_ALREADY_CREATED_PASSTHROUGH_FB = -1000118001,
// Provided by XR_FB_passthrough
XR_ERROR_FEATURE_REQUIRED_PASSTHROUGH_FB = -1000118002,
// Provided by XR_FB_passthrough
XR_ERROR_NOT_PERMITTED_PASSTHROUGH_FB = -1000118003,
// Provided by XR_FB_passthrough
XR_ERROR_INSUFFICIENT_RESOURCES_PASSTHROUGH_FB = -1000118004,
// Provided by XR_FB_passthrough
XR_ERROR_UNKNOWN_PASSTHROUGH_FB = -1000118050,
// Provided by XR_FB_render_model
XR_ERROR_RENDER_MODEL_KEY_INVALID_FB = -1000119000,
// Provided by XR_FB_render_model
XR_RENDER_MODEL_UNAVAILABLE_FB = 1000119020,
// Provided by XR_VARJO_marker_tracking
XR_ERROR_MARKER_NOT_TRACKED_VARJO = -1000124000,
// Provided by XR_VARJO_marker_tracking
XR_ERROR_MARKER_ID_INVALID_VARJO = -1000124001,
// Provided by XR_ML_marker_understanding
XR_ERROR_MARKER_DETECTOR_PERMISSION_DENIED_ML = -1000138000,
// Provided by XR_ML_marker_understanding
XR_ERROR_MARKER_DETECTOR_LOCATE_FAILED_ML = -1000138001,
// Provided by XR_ML_marker_understanding
XR_ERROR_MARKER_DETECTOR_INVALID_DATA_QUERY_ML = -1000138002,
// Provided by XR_ML_marker_understanding
XR_ERROR_MARKER_DETECTOR_INVALID_CREATE_INFO_ML = -1000138003,
// Provided by XR_ML_marker_understanding
XR_ERROR_MARKER_INVALID_ML = -1000138004,
// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_INCOMPATIBLE_ML = -1000139000,
// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_UNAVAILABLE_ML = -1000139001,
// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_FAIL_ML = -1000139002,

```

```

// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_IMPORT_EXPORT_PERMISSION_DENIED_ML = -1000139003,
// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_PERMISSION_DENIED_ML = -1000139004,
// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_ALREADY_EXISTS_ML = -1000139005,
// Provided by XR_ML_localization_map
XR_ERROR_LOCALIZATION_MAP_CANNOT_EXPORT_CLOUD_MAP_ML = -1000139006,
// Provided by XR_MSFT_spatial_anchor_persistence
XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT = -1000142001,
// Provided by XR_MSFT_spatial_anchor_persistence
XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT = -1000142002,
// Provided by XR_MSFT_scene_marker
XR_SCENE_MARKER_DATA_NOT_STRING_MSFT = 1000147000,
// Provided by XR_FB_spatial_entity_sharing
XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB = -1000169000,
// Provided by XR_FB_spatial_entity_sharing
XR_ERROR_SPACE_LOCALIZATION_FAILED_FB = -1000169001,
// Provided by XR_FB_spatial_entity_sharing
XR_ERROR_SPACE_NETWORK_TIMEOUT_FB = -1000169002,
// Provided by XR_FB_spatial_entity_sharing
XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB = -1000169003,
// Provided by XR_FB_spatial_entity_sharing
XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB = -1000169004,
// Provided by XR_META_passthrough_color_lut
XR_ERROR_PASSTHROUGH_COLOR_LUT_BUFFER_SIZE_MISMATCH_META = -1000266000,
// Provided by XR_META_environment_depth
XR_ENVIRONMENT_DEPTH_NOT_AVAILABLE_META = 1000291000,
// Provided by XR_QCOM_tracking_optimization_settings
XR_ERROR_HINT_ALREADY_SET_QCOM = -1000306000,
// Provided by XR_HTC_anchor
XR_ERROR_NOT_AN_ANCHOR_HTC = -1000319000,
// Provided by XR_EXT_plane_detection
XR_ERROR_SPACE_NOT_LOCATABLE_EXT = -1000429000,
// Provided by XR_EXT_plane_detection
XR_ERROR_PLANE_DETECTION_PERMISSION_DENIED_EXT = -1000429001,
// Provided by XR_EXT_future
XR_ERROR_FUTURE_PENDING_EXT = -1000469001,
// Provided by XR_EXT_future
XR_ERROR_FUTURE_INVALID_EXT = -1000469002,
// Provided by XR_KHR_maintenance1
XR_ERROR_EXTENSION_DEPENDENCY_NOT_ENABLED_KHR =
XR_ERROR_EXTENSION_DEPENDENCY_NOT_ENABLED,
// Provided by XR_KHR_maintenance1
XR_ERROR_PERMISSION_INSUFFICIENT_KHR = XR_ERROR_PERMISSION_INSUFFICIENT,
XR_RESULT_MAX_ENUM = 0x7FFFFFFF
} XrResult;

```

All return codes in the API are reported via [XrResult](#) return values.

The following are common suffixes shared across many of the return codes:

- **\_INVALID**: The specified handle, atom, or value is formatted incorrectly, or the specified handle was never created or has been destroyed.
- **\_UNSUPPORTED**: The specified handle, atom, enumerant, or value is formatted correctly but cannot be used for the lifetime of this function's parent handle.
- **\_UNAVAILABLE**: The specified handle, atom, enumerant, or value is supported by the handle taken by this function, but is not usable at this moment.

## Success Codes

| Enum   | Description   |
|--|---|
| <code>XR_SUCCESS</code>                              | Function successfully completed.  |
| <code>XR_TIMEOUT_EXPIRED</code>                      | The specified timeout time occurred before the operation could complete.  |
| <code>XR_SESSION_LOSS_PENDING</code>                 | The session will be lost soon.  |
| <code>XR_EVENT_UNAVAILABLE</code>                    | No event was available.   |
| <code>XR_SPACE_BOUNDS_UNAVAILABLE</code>             | The space's bounds are not known at the moment.   |
| <code>XR_SESSION_NOT_FOCUSED</code>                  | The session is not in the focused state.  |
| <code>XR_FRAME_DISCARDED</code>                      | A frame has been discarded from composition.  |
| <code>XR_RENDER_MODEL_UNAVAILABLE_FB</code>          | The model is unavailable. (Added by the <a href="#">XR_FB_render_model</a> extension)                                       |
| <code>XR_SCENE_MARKER_DATA_NOT_STRING_MSFT</code>    | Marker does not encode a string. (Added by the <a href="#">XR_MSFT_scene_marker</a> extension)                              |
| <code>XR_ENVIRONMENT_DEPTH_NOT_AVAILABLE_META</code> | Warning: The requested depth image is not yet available. (Added by the <a href="#">XR_META_environment_depth</a> extension) |

## Error Codes

| Enum                                     | Description   |
|--|---|
| <code>XR_ERROR_VALIDATION_FAILURE</code> | The function usage was invalid in some way.   |
| <code>XR_ERROR_RUNTIME_FAILURE</code>    | The runtime failed to handle the function in an unexpected way that is not covered by another error result. |
| <code>XR_ERROR_OUT_OF_MEMORY</code>      | A memory allocation has failed.   |

| Enum   | Description  |
|--|--|
| <code>XR_ERROR_API_VERSION_UNSUPPORTED</code>      | The runtime does not support the requested API version.  |
| <code>XR_ERROR_INITIALIZATION_FAILED</code>        | Initialization of object could not be completed.   |
| <code>XR_ERROR_FUNCTION_UNSUPPORTED</code>         | The requested function was not found or is otherwise unsupported.  |
| <code>XR_ERROR_FEATURE_UNSUPPORTED</code>          | The requested feature is not supported.  |
| <code>XR_ERROR_EXTENSION_NOT_PRESENT</code>        | A requested extension is not supported.  |
| <code>XR_ERROR_LIMIT_REACHED</code>                | The runtime supports no more of the requested resource.  |
| <code>XR_ERROR_SIZE_INSUFFICIENT</code>            | The supplied size was smaller than required.   |
| <code>XR_ERROR_HANDLE_INVALID</code>               | A supplied object handle was invalid.  |
| <code>XR_ERROR_INSTANCE_LOST</code>                | The <code>XrInstance</code> was lost or could not be found. It will need to be destroyed and optionally recreated. |
| <code>XR_ERROR_SESSION_RUNNING</code>              | The session <b>is already running</b> .  |
| <code>XR_ERROR_SESSION_NOT_RUNNING</code>          | The session <b>is not yet running</b> .  |
| <code>XR_ERROR_SESSION_LOST</code>                 | The <code>XrSession</code> was lost. It will need to be destroyed and optionally recreated.                        |
| <code>XR_ERROR_SYSTEM_INVALID</code>               | The provided <code>XrSystemId</code> was invalid.  |
| <code>XR_ERROR_PATH_INVALID</code>                 | The provided <code>XrPath</code> was not valid.  |
| <code>XR_ERROR_PATH_COUNT_EXCEEDED</code>          | The maximum number of supported semantic paths has been reached.   |
| <code>XR_ERROR_PATH_FORMAT_INVALID</code>          | The semantic path character format is invalid.   |
| <code>XR_ERROR_PATH_UNSUPPORTED</code>             | The semantic path is unsupported.  |
| <code>XR_ERROR_LAYER_INVALID</code>                | The layer was NULL or otherwise invalid.   |
| <code>XR_ERROR_LAYER_LIMIT_EXCEEDED</code>         | The number of specified layers is greater than the supported number.   |
| <code>XR_ERROR_SWAPCHAIN_RECT_INVALID</code>       | The image rect was negatively sized or otherwise invalid.  |
| <code>XR_ERROR_SWAPCHAIN_FORMAT_UNSUPPORTED</code> | The image format is not supported by the runtime or platform.  |
| <code>XR_ERROR_ACTION_TYPE_MISMATCH</code>         | The API used to retrieve an action's state does not match the action's type.                                       |
| <code>XR_ERROR_SESSION_NOT_READY</code>            | The session is not in the ready state.   |



| Enum  | Description  |
|---|--|
| <code>XR_ERROR_SESSION_NOT_STOPPING</code>                | The session is not in the stopping state.  |
| <code>XR_ERROR_TIME_INVALID</code>                        | The provided <code>XrTime</code> was zero, negative, or out of range.  |
| <code>XR_ERROR_REFERENCE_SPACE_UNSUPPORTED</code>         | The specified reference space is not supported by the runtime or system.   |
| <code>XR_ERROR_FILE_ACCESS_ERROR</code>                   | The file could not be accessed.  |
| <code>XR_ERROR_FILE_CONTENTS_INVALID</code>               | The file's contents were invalid.  |
| <code>XR_ERROR_FORM_FACTOR_UNSUPPORTED</code>             | The specified form factor is not supported by the current runtime or platform.   |
| <code>XR_ERROR_FORM_FACTOR_UNAVAILABLE</code>             | The specified form factor is supported, but the device is currently not available, e.g. not plugged in or powered off.                     |
| <code>XR_ERROR_API_LAYER_NOT_PRESENT</code>               | A requested API layer is not present or could not be loaded.   |
| <code>XR_ERROR_CALL_ORDER_INVALID</code>                  | The call was made without having made a previously required call.  |
| <code>XR_ERROR_GRAPHICS_DEVICE_INVALID</code>             | The given graphics device is not in a valid state. The graphics device could be lost or initialized without meeting graphics requirements. |
| <code>XR_ERROR_POSE_INVALID</code>                        | The supplied pose was invalid with respect to the requirements.  |
| <code>XR_ERROR_INDEX_OUT_OF_RANGE</code>                  | The supplied index was outside the range of valid indices.   |
| <code>XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED</code> | The specified view configuration type is not supported by the runtime or platform.   |
| <code>XR_ERROR_ENVIRONMENT_BLEND_MODE_UNSUPPORTED</code>  | The specified environment blend mode is not supported by the runtime or platform.  |
| <code>XR_ERROR_NAME_DUPLICATED</code>                     | The name provided was a duplicate of an already-existing resource.   |
| <code>XR_ERROR_NAME_INVALID</code>                        | The name provided was invalid.   |
| <code>XR_ERROR_ACTIONSET_NOT_ATTACHED</code>              | A referenced action set is not attached to the session.  |
| <code>XR_ERROR_ACTIONSETS_ALREADY_ATTACHED</code>         | The session already has attached action sets.  |
| <code>XR_ERROR_LOCALIZED_NAME_DUPLICATED</code>           | The localized name provided was a duplicate of an already-existing resource.   |

| Enum   | Description  |
|--|--|
| <code>XR_ERROR_LOCALIZED_NAME_INVALID</code>                             | The localized name provided was invalid.   |
| <code>XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING</code>                 | The <code>xrGetGraphicsRequirements*</code> call was not made before calling <code>xrCreateSession</code> .  |
| <code>XR_ERROR_RUNTIME_UNAVAILABLE</code>                                | The loader was unable to find or load a runtime.   |
| <code>XR_ERROR_EXTENSION_DEPENDENCY_NOT_ENABLED</code>                   | One or more of the extensions being enabled has dependency on extensions that are not enabled.   |
| <code>XR_ERROR_PERMISSION_INSUFFICIENT</code>                            | Insufficient permissions. This error is included for use by vendor extensions. The precise definition of <code>XR_ERROR_PERMISSION_INSUFFICIENT</code> and actions possible by the developer or user to resolve it can vary by platform, extension or function. The developer should refer to the documentation of the function that returned the error code and extension it was defined. |
| <code>XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR</code>             | <code>xrSetAndroidApplicationThreadKHR</code> failed as thread id is invalid. (Added by the <a href="#">XR_KHR_android_thread_settings</a> extension)  |
| <code>XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR</code>                | <code>xrSetAndroidApplicationThreadKHR</code> failed setting the thread attributes/priority. (Added by the <a href="#">XR_KHR_android_thread_settings</a> extension)   |
| <code>XR_ERROR_CREATE_SPATIAL_ANCHOR_FAILED_MSFT</code>                  | Spatial anchor could not be created at that location. (Added by the <a href="#">XR_MSFT_spatial_anchor</a> extension)  |
| <code>XR_ERROR_SECONDARY_VIEW_CONFIGURATION_TYPE_NOT_ENABLED_MSFT</code> | The secondary view configuration was not enabled when creating the session. (Added by the <a href="#">XR_MSFT_secondary_view_configuration</a> extension)  |
| <code>XR_ERROR_CONTROLLER_MODEL_KEY_INVALID_MSFT</code>                  | The controller model key is invalid. (Added by the <a href="#">XR_MSFT_controller_model</a> extension)   |
| <code>XR_ERROR_REPROJECTION_MODE_UNSUPPORTED_MSFT</code>                 | The reprojection mode is not supported. (Added by the <a href="#">XR_MSFT_composition_layer_reprojection</a> extension)  |
| <code>XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT</code>               | Compute new scene not completed. (Added by the <a href="#">XR_MSFT_scene_understanding</a> extension)  |
| <code>XR_ERROR_SCENE_COMPONENT_ID_INVALID_MSFT</code>                    | Scene component id invalid. (Added by the <a href="#">XR_MSFT_scene_understanding</a> extension)   |
| <code>XR_ERROR_SCENE_COMPONENT_TYPE_MISMATCH_MSFT</code>                 | Scene component type mismatch. (Added by the <a href="#">XR_MSFT_scene_understanding</a> extension)  |

| Enum  | Description  |
|---|--|
| <code>XR_ERROR_SCENE_MESH_BUFFER_ID_INVALID_MSFT</code>       | Scene mesh buffer id invalid. (Added by the <a href="#">XR_MSFT_scene_understanding</a> extension)   |
| <code>XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT</code> | Scene compute feature incompatible. (Added by the <a href="#">XR_MSFT_scene_understanding</a> extension)   |
| <code>XR_ERROR_SCENE_COMPUTE_CONSISTENCY_MISMATCH_MSFT</code> | Scene compute consistency mismatch. (Added by the <a href="#">XR_MSFT_scene_understanding</a> extension)   |
| <code>XR_ERROR_DISPLAY_REFRESH_RATE_UNSUPPORTED_FB</code>     | The display refresh rate is not supported by the platform. (Added by the <a href="#">XR_FB_display_refresh_rate</a> extension)                                       |
| <code>XR_ERROR_COLOR_SPACE_UNSUPPORTED_FB</code>              | The color space is not supported by the runtime. (Added by the <a href="#">XR_FB_color_space</a> extension)  |
| <code>XR_ERROR_SPACE_COMPONENT_NOT_SUPPORTED_FB</code>        | The component type is not supported for this space. (Added by the <a href="#">XR_FB_spatial_entity</a> extension)  |
| <code>XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB</code>          | The required component is not enabled for this space. (Added by the <a href="#">XR_FB_spatial_entity</a> extension)  |
| <code>XR_ERROR_SPACE_COMPONENT_STATUS_PENDING_FB</code>       | A request to set the component's status is currently pending. (Added by the <a href="#">XR_FB_spatial_entity</a> extension)  |
| <code>XR_ERROR_SPACE_COMPONENT_STATUS_ALREADY_SET_FB</code>   | The component is already set to the requested value. (Added by the <a href="#">XR_FB_spatial_entity</a> extension)   |
| <code>XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB</code>         | The object state is unexpected for the issued command. (Added by the <a href="#">XR_FB_passthrough</a> extension)  |
| <code>XR_ERROR_FEATURE_ALREADY_CREATED_PASSTHROUGH_FB</code>  | Trying to create an MR feature when one was already created and only one instance is allowed. (Added by the <a href="#">XR_FB_passthrough</a> extension)             |
| <code>XR_ERROR_FEATURE_REQUIRED_PASSTHROUGH_FB</code>         | Requested functionality requires a feature to be created first. (Added by the <a href="#">XR_FB_passthrough</a> extension)   |
| <code>XR_ERROR_NOT_PERMITTED_PASSTHROUGH_FB</code>            | Requested functionality is not permitted - application is not allowed to perform the requested operation. (Added by the <a href="#">XR_FB_passthrough</a> extension) |

| Enum   | Description   |
|--|---|
| <code>XR_ERROR_INSUFFICIENT_RESOURCES_PASSTHROUGH_FB</code>  | There were insufficient resources available to perform an operation. (Added by the <a href="#">XR_FB_passthrough</a> extension)   |
| <code>XR_ERROR_UNKNOWN_PASSTHROUGH_FB</code>                 | Unknown Passthrough error (no further details provided). (Added by the <a href="#">XR_FB_passthrough</a> extension)   |
| <code>XR_ERROR_RENDER_MODEL_KEY_INVALID_FB</code>            | The model key is invalid. (Added by the <a href="#">XR_FB_render_model</a> extension)   |
| <code>XR_ERROR_MARKER_NOT_TRACKED_VARJO</code>               | Marker tracking is disabled or the specified marker is not currently tracked. (Added by the <a href="#">XR_VARJO_marker_tracking</a> extension)   |
| <code>XR_ERROR_MARKER_ID_INVALID_VARJO</code>                | The specified marker ID is not valid. (Added by the <a href="#">XR_VARJO_marker_tracking</a> extension)   |
| <code>XR_ERROR_MARKER_DETECTOR_PERMISSION_DENIED_ML</code>   | The <code>com.magicleap.permission.MARKER_TRACKING</code> permission was denied. (Added by the <a href="#">XR_ML_marker_understanding</a> extension)  |
| <code>XR_ERROR_MARKER_DETECTOR_LOCATE_FAILED_ML</code>       | The specified marker could not be located spatially. (Added by the <a href="#">XR_ML_marker_understanding</a> extension)  |
| <code>XR_ERROR_MARKER_DETECTOR_INVALID_DATA_QUERY_ML</code>  | The marker queried does not contain data of the requested type. (Added by the <a href="#">XR_ML_marker_understanding</a> extension)   |
| <code>XR_ERROR_MARKER_DETECTOR_INVALID_CREATE_INFO_ML</code> | <code>createInfo</code> contains mutually exclusive parameters, such as setting <code>XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_APRIL_TAG_ML</code> with <code>XR_MARKER_TYPE_ARUCO_ML</code> . (Added by the <a href="#">XR_ML_marker_understanding</a> extension) |
| <code>XR_ERROR_MARKER_INVALID_ML</code>                      | The marker id passed to the function was invalid. (Added by the <a href="#">XR_ML_marker_understanding</a> extension)   |
| <code>XR_ERROR_LOCALIZATION_MAP_INCOMPATIBLE_ML</code>       | The localization map being imported is not compatible with current OS or mode. (Added by the <a href="#">XR_ML_localization_map</a> extension)  |
| <code>XR_ERROR_LOCALIZATION_MAP_UNAVAILABLE_ML</code>        | The localization map requested is not available. (Added by the <a href="#">XR_ML_localization_map</a> extension)  |

| Enum  | Description   |
|---|---|
| <code>XR_ERROR_LOCALIZATION_MAP_FAIL_ML</code>                            | The map localization service failed to fulfill the request, retry later. (Added by the <a href="#">XR_ML_localization_map</a> extension)                      |
| <code>XR_ERROR_LOCALIZATION_MAP_IMPORT_EXPORT_PERMISSION_DENIED_ML</code> | The <code>com.magicleap.permission.SPACE_IMPORT_EXPORT</code> permission was denied. (Added by the <a href="#">XR_ML_localization_map</a> extension)          |
| <code>XR_ERROR_LOCALIZATION_MAP_PERMISSION_DENIED_ML</code>               | The <code>com.magicleap.permission.SPACE_MANAGER</code> permission was denied. (Added by the <a href="#">XR_ML_localization_map</a> extension)                |
| <code>XR_ERROR_LOCALIZATION_MAP_ALREADY_EXISTS_ML</code>                  | The map being imported already exists in the system. (Added by the <a href="#">XR_ML_localization_map</a> extension)  |
| <code>XR_ERROR_LOCALIZATION_MAP_CANNOT_EXPORT_CLOUD_MAP_ML</code>         | The map localization service cannot export cloud based maps. (Added by the <a href="#">XR_ML_localization_map</a> extension)                                  |
| <code>XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT</code>                  | A spatial anchor was not found associated with the spatial anchor name provided (Added by the <a href="#">XR_MSFT_spatial_anchor_persistence</a> extension)   |
| <code>XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT</code>                    | The spatial anchor name provided was not valid (Added by the <a href="#">XR_MSFT_spatial_anchor_persistence</a> extension)                                    |
| <code>XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB</code>                       | Anchor import from cloud or export from device failed. (Added by the <a href="#">XR_FB_spatial_entity_sharing</a> extension)                                  |
| <code>XR_ERROR_SPACE_LOCALIZATION_FAILED_FB</code>                        | Anchors were downloaded from the cloud but failed to be imported/aligned on the device. (Added by the <a href="#">XR_FB_spatial_entity_sharing</a> extension) |
| <code>XR_ERROR_SPACE_NETWORK_TIMEOUT_FB</code>                            | Timeout occurred while waiting for network request to complete. (Added by the <a href="#">XR_FB_spatial_entity_sharing</a> extension)                         |
| <code>XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB</code>                     | The network request failed. (Added by the <a href="#">XR_FB_spatial_entity_sharing</a> extension)   |
| <code>XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB</code>                     | Cloud storage is required for this operation but is currently disabled. (Added by the <a href="#">XR_FB_spatial_entity_sharing</a> extension)                 |

| Enum  | Description   |
|---|---|
| <code>XR_ERROR_PASSTHROUGH_COLOR_LUT_BUFFER_SIZE_MISMATCH_META</code> | The provided data buffer did not match the required size. (Added by the <a href="#">XR_META_passthrough_color_lut</a> extension)          |
| <code>XR_ERROR_HINT_ALREADY_SET_QCOM</code>                           | Tracking optimization hint is already set for the domain. (Added by the <a href="#">XR_QCOM_tracking_optimization_settings</a> extension) |
| <code>XR_ERROR_NOT_AN_ANCHOR_HTC</code>                               | The provided space is valid but not an anchor. (Added by the <a href="#">XR_HTC_anchor</a> extension)                                     |
| <code>XR_ERROR_SPACE_NOT_LOCATABLE_EXT</code>                         | The space passed to the function was not locatable. (Added by the <a href="#">XR_EXT_plane_detection</a> extension)                       |
| <code>XR_ERROR_PLANE_DETECTION_PERMISSION_DENIED_EXT</code>           | The permission for this resource was not granted. (Added by the <a href="#">XR_EXT_plane_detection</a> extension)                         |
| <code>XR_ERROR_FUTURE_PENDING_EXT</code>                              | Returned by completion function to indicate future is not ready. (Added by the <a href="#">XR_EXT_future</a> extension)                   |
| <code>XR_ERROR_FUTURE_INVALID_EXT</code>                              | Returned by completion function to indicate future is not valid. (Added by the <a href="#">XR_EXT_future</a> extension)                   |

### 2.10.1. Convenience Macros

```
// Provided by XR_VERSION_1_0
#define XR_SUCCEEDED(result) ((result) >= 0)
```

A convenience macro that **can** be used to test if a function succeeded. Note that this evaluates to true for all success codes, including a qualified success such as `XR_FRAME_DISCARDED`.

```
// Provided by XR_VERSION_1_0
#define XR_FAILED(result) ((result) < 0)
```

A convenience macro that **can** be used to test if a function has failed in some way. It evaluates to true for all failure codes.

```
// Provided by XR_VERSION_1_0
#define XR_UNQUALIFIED_SUCCESS(result) ((result) == 0)
```

A convenience macro that can be used to test a function's failure. The `XR_UNQUALIFIED_SUCCESS` macro evaluates to true exclusively when the provided `XrResult` is equal to `XR_SUCCESS (0)`.

## 2.10.2. Validation

Except as noted below or in individual API specifications, valid API usage **may** be required by the runtime. Runtimes **may** choose to validate some API usage and return an appropriate error code.

Application developers **should** use validation layers to catch and eliminate errors during development. Once validated, applications **should** not enable validation layers by default.

If a function returns a run time error, unless otherwise specified any output parameters will have undefined contents, except that if the output parameter is a structure with `type` and `next` fields, those fields will be unmodified. Any output structures chained from `next` will also have undefined contents, except that the `type` and `next` will be unmodified.

Unless otherwise specified, errors do not affect existing OpenXR objects. Objects that have already been successfully created **may** still be used by the application.

`XrResult` code returns **may** be added to a given function in future versions of the specification. Runtimes **must** return only `XrResult` codes from the set documented for the given application API version.

Runtimes **must** ensure that incorrect usage by an application does not affect the integrity of the operating system, the API implementation, or other API client applications in the system, and does not allow one application to access data belonging to another application.

## 2.11. Handles

Objects which are allocated by the runtime on behalf of applications are represented by handles. Handles are opaque identifiers for objects whose lifetime is controlled by applications via the create and destroy functions. Example handle types include `XrInstance`, `XrSession`, and `XrSwapchain`. Handles which have not been destroyed are unique for a given application process, but **may** be reused after being destroyed. Unless otherwise specified, a successful handle creation function call returns a new unique handle. Unless otherwise specified, handles are implicitly destroyed when their parent handle is destroyed. Applications **may** destroy handles explicitly before the parent handle is destroyed, and **should** do so if no longer needed, in order to conserve resources. Runtimes **may** detect `XR_NULL_HANDLE` and other invalid handles passed where a valid handle is required and return `XR_ERROR_HANDLE_INVALID`. However, runtimes are not required to do so unless otherwise specified, and so use of any invalid handle **may** result in undefined behavior. When a function has an **optional**

handle parameter, `XR_NULL_HANDLE` **must** be passed by the application if it does not pass a valid handle.

All functions that take a handle parameter **may** return `XR_ERROR_HANDLE_INVALID`.

Handles form a hierarchy in which child handles fall under the validity and lifetime of parent handles. For example, to create an `XrSwapchain` handle, applications must call `xrCreateSwapchain` and pass an `XrSession` handle. Thus `XrSwapchain` is a child handle of `XrSession`.

## 2.12. Object Handle Types

The type of an object handle used in a function is usually determined by the specification of that function, as discussed in [Valid Usage for Object Handles](#). However, some functions accept or return object handle parameters where the type of the object handle is unknown at execution time and is not specified in the description of the function itself. For these functions, the `XrObjectType` **may** be used to explicitly specify the type of a handle.

For example, an information-gathering or debugging mechanism implemented in a runtime extension or API layer extension **may** return a list of object handles that are generated by the mechanism's operation. The same mechanism **may** also return a parallel list of object handle types that allow the recipient of this information to easily determine the types of the handles.

In general, anywhere an object handle of more than one type can occur, the object handle type **may** be provided to indicate its type.

```
// Provided by XR_VERSION_1_0
typedef enum XrObjectType {
    XR_OBJECT_TYPE_UNKNOWN = 0,
    XR_OBJECT_TYPE_INSTANCE = 1,
    XR_OBJECT_TYPE_SESSION = 2,
    XR_OBJECT_TYPE_SWAPCHAIN = 3,
    XR_OBJECT_TYPE_SPACE = 4,
    XR_OBJECT_TYPE_ACTION_SET = 5,
    XR_OBJECT_TYPE_ACTION = 6,
// Provided by XR_EXT_debug_utils
    XR_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT = 1000019000,
// Provided by XR_MSFT_spatial_anchor
    XR_OBJECT_TYPE_SPATIAL_ANCHOR_MSFT = 1000039000,
// Provided by XR_MSFT_spatial_graph_bridge
    XR_OBJECT_TYPE_SPATIAL_GRAPH_NODE_BINDING_MSFT = 1000049000,
// Provided by XR_EXT_hand_tracking
    XR_OBJECT_TYPE_HAND_TRACKER_EXT = 1000051000,
// Provided by XR_FB_body_tracking
    XR_OBJECT_TYPE_BODY_TRACKER_FB = 1000076000,
```



```

// Provided by XR_MSFT_scene_understanding
XR_OBJECT_TYPE_SCENE_OBSERVER_MSFT = 1000097000,
// Provided by XR_MSFT_scene_understanding
XR_OBJECT_TYPE_SCENE_MSFT = 1000097001,
// Provided by XR_HTC_facial_tracking
XR_OBJECT_TYPE_FACIAL_TRACKER_HTC = 1000104000,
// Provided by XR_FB_foveation
XR_OBJECT_TYPE_FOVEATION_PROFILE_FB = 1000114000,
// Provided by XR_FB_triangle_mesh
XR_OBJECT_TYPE_TRIANGLE_MESH_FB = 1000117000,
// Provided by XR_FB_passthrough
XR_OBJECT_TYPE_PASSTHROUGH_FB = 1000118000,
// Provided by XR_FB_passthrough
XR_OBJECT_TYPE_PASSTHROUGH_LAYER_FB = 1000118002,
// Provided by XR_FB_passthrough
XR_OBJECT_TYPE_GEOMETRY_INSTANCE_FB = 1000118004,
// Provided by XR_ML_marker_understanding
XR_OBJECT_TYPE_MARKER_DETECTOR_ML = 1000138000,
// Provided by XR_ML_localization_map
XR_OBJECT_TYPE_EXPORTED_LOCALIZATION_MAP_ML = 1000139000,
// Provided by XR_MSFT_spatial_anchor_persistence
XR_OBJECT_TYPE_SPATIAL_ANCHOR_STORE_CONNECTION_MSFT = 1000142000,
// Provided by XR_FB_face_tracking
XR_OBJECT_TYPE_FACE_TRACKER_FB = 1000201000,
// Provided by XR_FB_eye_tracking_social
XR_OBJECT_TYPE_EYE_TRACKER_FB = 1000202000,
// Provided by XR_META_virtual_keyboard
XR_OBJECT_TYPE_VIRTUAL_KEYBOARD_META = 1000219000,
// Provided by XR_FB_spatial_entity_user
XR_OBJECT_TYPE_SPACE_USER_FB = 1000241000,
// Provided by XR_META_passthrough_color_lut
XR_OBJECT_TYPE_PASSTHROUGH_COLOR_LUT_META = 1000266000,
// Provided by XR_FB_face_tracking2
XR_OBJECT_TYPE_FACE_TRACKER2_FB = 1000287012,
// Provided by XR_META_environment_depth
XR_OBJECT_TYPE_ENVIRONMENT_DEPTH_PROVIDER_META = 1000291000,
// Provided by XR_META_environment_depth
XR_OBJECT_TYPE_ENVIRONMENT_DEPTH_SWAPCHAIN_META = 1000291001,
// Provided by XR_HTC_passthrough
XR_OBJECT_TYPE_PASSTHROUGH_HTC = 1000317000,
// Provided by XR_EXT_plane_detection
XR_OBJECT_TYPE_PLANE_DETECTOR_EXT = 1000429000,
XR_OBJECT_TYPE_MAX_ENUM = 0x7FFFFFFF
} XrObjectType;

```

The `XrObjectType` enumeration defines values, each of which corresponds to a specific OpenXR handle type. These values **can** be used to associate debug information with a particular type of object through

one or more extensions.

The following table defines [XrObjectType](#) and OpenXR Handle relationships in the core specification:

| <a href="#">XrObjectType</a>              | OpenXR Handle Type          |
|---|-----------------------------|
| <a href="#">XR_OBJECT_TYPE_UNKNOWN</a>    | Unknown/Undefined Handle    |
| <a href="#">XR_OBJECT_TYPE_INSTANCE</a>   | <a href="#">XrInstance</a>  |
| <a href="#">XR_OBJECT_TYPE_SESSION</a>    | <a href="#">XrSession</a>   |
| <a href="#">XR_OBJECT_TYPE_SWAPCHAIN</a>  | <a href="#">XrSwapchain</a> |
| <a href="#">XR_OBJECT_TYPE_SPACE</a>      | <a href="#">XrSpace</a>     |
| <a href="#">XR_OBJECT_TYPE_ACTION_SET</a> | <a href="#">XrActionSet</a> |
| <a href="#">XR_OBJECT_TYPE_ACTION</a>     | <a href="#">XrAction</a>    |

## 2.13. Buffer Size Parameters

Functions with input/output buffer parameters take on either parameter form or structure form, as in one of the following examples, with the element type being `float` in this case:

Parameter form:

```
XrResult xrFunction(uint32_t elementCapacityInput, uint32_t* elementCountOutput, float* elements);
```

Structure form:

```
XrResult xrFunction(XrBuffer* buffer);

struct XrBuffer {
    uint32_t          elementCapacityInput;
    uint32_t          elementCountOutput;
    float*            elements;
};
```

A "two-call idiom" **should** be employed by the application, first calling `xrFunction` (with a valid `elementCountOutput` pointer if in parameter form), but passing `NULL` as `elements` and `0` as `elementCapacityInput`, to retrieve the required buffer size as number of elements (number of floats in this example). After allocating a buffer at least as large as `elementCountOutput` (in a structure) or the value pointed to by `elementCountOutput` (as parameters), a pointer to the allocated buffer **should** be passed as `elements`, along with the buffer's length in `elementCapacityInput`, to a second call to `xrFunction` to perform the retrieval of the data. If the element type of `elements` is a structure with `type` and `next` fields, the application **must** set the `type` to the correct value, and **must** set `next` to a valid value. A valid

value for `next` is generally either `NULL` or another structure with related data, in which `type` and `next` are also valid, recursively. (See [Valid Usage for Structure Pointer Chains](#) for details.)

In the following discussion, "set `elementCountOutput`" should be interpreted as "set the value pointed to by `elementCountOutput`" in parameter form and "set the value of `elementCountOutput`" in struct form. These functions have the following behavior with respect to the array/buffer and its size parameters:

### Buffer Size Parameter Behavior

- The `elementCapacityInput` and `elementCountOutput` arguments precede the array to which they refer, in argument order.
- `elementCapacityInput` specifies the capacity in number of elements of the buffer to be written, or `0` to indicate a request for the required buffer size.
- Independent of `elementCapacityInput` or `elements` parameters, the application **must** pass a valid pointer for `elementCountOutput` if the function uses parameter form.
- Independent of `elementCapacityInput` or `elements` parameters, the function sets `elementCountOutput`.
- The application **may** pass `0` for the `elementCapacityInput` parameter, to indicate a request for the required array size. That is, passing a capacity of `0` does not return `XR_ERROR_SIZE_INSUFFICIENT`. In this case, the following two points apply.
  - The function **must** set `elementCountOutput` to the required size in number of elements.
  - The `elements` parameter is ignored (any value passed is considered valid usage).
- If the `elementCapacityInput` is non-zero but less than required, the function **must**: set `elementCountOutput` to the required capacity, and **must** return `XR_ERROR_SIZE_INSUFFICIENT`. After the function returns, the data in the array `elements` is undefined.
- If the `elementCapacityInput` is non-zero and the function returns successfully, the function sets `elementCountOutput` to the count of the elements that have been written to `elements`.
- If the function fails for reasons unrelated to the element array capacity, the contents of the values of (or pointed to by) `elementCountOutput` and `elements` are undefined.
- For clarity, if the element array refers to a string (`element` is of type `char*`), `elementCapacityInput` and `elementCountOutput` refer to the string `strlen` plus `1` for a `NULL` terminator.

Some functions have a given `elementCapacityInput` and `elementCountOutput` associated with more than one element array (i.e. parallel arrays). In this case, the capacity/count and all its associated arrays will share a common prefix. All of the preceding general requirements continue to apply.

Some functions fill multiple element arrays of varying sizes in one call. For these functions, the `elementCapacityInput`, `elementCountOutput`, and `elements` array parameters or fields are repeated with different prefixes. In this case, all of the preceding general requirements still apply, with these

additional requirements:

- If the application sets **any** `elementCapacityInput` parameter or field to `0`, the runtime **must** treat **all** `elementCapacityInput` values as if they were set to `0`.
- If all `elementCapacityInput` values are non-zero but **any** is insufficient to fit all elements of its corresponding array, the runtime **must** return `XR_ERROR_SIZE_INSUFFICIENT`. As in the case of the single array, the data in all arrays is undefined when `XR_ERROR_SIZE_INSUFFICIENT` is returned.

## 2.14. Time

Time is represented by a 64-bit signed integer representing nanoseconds (`XrTime`). The passage of time **must** be monotonic and not real-time (i.e. wall clock time). Thus the time is always increasing at a constant rate and is unaffected by clock changes, time zones, daylight savings, etc.

### 2.14.1. XrTime

```
typedef int64_t XrTime;
```

`XrTime` is a base value type that represents time as a signed 64-bit integer, representing the monotonically-increasing count of nanoseconds that have elapsed since a runtime-chosen epoch. `XrTime` always represents the time elapsed since that constant epoch, rather than a duration or a time point relative to some moving epoch such as vsync time, etc. Durations are instead represented by `XrDuration`.

A single runtime **must** use the same epoch for all simultaneous applications. Time **must** be represented the same regardless of multiple processors or threads present in the system.

The period precision of time reported by the runtime is runtime-dependent, and **may** change. One nanosecond is the finest possible period precision. A runtime **may**, for example, report time progression with only microsecond-level granularity.

Time **must** not be assumed to correspond to a system clock time.

Unless specified otherwise, zero or a negative value is not a valid `XrTime`, and related functions **must** return error `XR_ERROR_TIME_INVALID`. Applications **must** not initialize such `XrTime` fields to a zero value. Instead, applications **should** always assign `XrTime` fields to the meaningful point in time they are choosing to reason about, such as a frame's predicted display time, or an action's last change time.

The behavior of a runtime is undefined when time overflows beyond the maximum positive value that can be represented by an `XrTime`. Runtimes **should** choose an epoch that minimizes the chance of overflow. Runtimes **should** also choose an epoch that minimizes the chance of underflow below 0 for applications performing a reasonable amount of historical pose lookback. For example, if the runtime

chooses an epoch relative to its startup time, it **should** push the epoch into the past by enough time to avoid applications performing reasonable pose lookback from reaching a negative [XrTime](#) value.

An application cannot assume that the system's clock and the runtime's clock will maintain a constant relationship across frames and **should** avoid storing such an offset, as this may cause time drift. Applications **should** instead always use time interop functions to convert a relevant time point across the system's clock and the runtime's clock using extensions, for example, [XR\\_KHR\\_win32\\_convert\\_performance\\_counter\\_time](#) or [XR\\_KHR\\_convert\\_timespec\\_time](#).

## 2.15. Duration

Duration refers to an elapsed period of time, as opposed to an absolute timepoint.

### 2.15.1. XrDuration

```
typedef int64_t XrDuration;
```

The difference between two timepoints is a duration, and thus the difference between two [XrTime](#) values is an [XrDuration](#) value. [XrDuration](#) is a base value type that represents duration as a signed 64-bit integer, representing the signed number of nanoseconds between two timepoints.

Functions that refer to durations use [XrDuration](#) as opposed to [XrTime](#). When an [XrDuration](#) is used as a timeout parameter, the constants [XR\\_NO\\_DURATION](#) and [XR\\_INFINITE\\_DURATION](#) have special meaning. A timeout with a duration that refers to the past (that is, a negative duration) **must** be interpreted as a timeout of [XR\\_NO\\_DURATION](#).

The interpretation of zero and negative durations in non-timeout uses is specified along with each such use.

```
// Provided by XR_VERSION_1_0
#define XR_NO_DURATION 0
```

For the case of timeout durations, [XR\\_NO\\_DURATION](#) **can** be used to indicate that the timeout is immediate.

```
// Provided by XR_VERSION_1_0
#define XR_INFINITE_DURATION 0x7fffffffffffffffLL
```

[XR\\_INFINITE\\_DURATION](#) is a special value that **can** be used to indicate that the timeout never occurs.

## 2.16. Prediction Time Limits

Some functions involve prediction. For example, [xrLocateViews](#) accepts a display time for which to return the resulting data. Prediction times provided by applications may refer to time in the past or the future. Times in the past **may** be interpolated historical data. Runtimes have different practical limits with respect to how far forward or backward prediction times can be accurate. There is no prescribed forward limit the application can successfully request predictions for, though predictions may become less accurate as they get farther into the future. 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. Requested times predating this time window, or requested times predating the earliest received pose, **may** result in a best effort data whose accuracy reduced or unspecified.

## 2.17. Colors

The [XrColor3f](#) structure is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrColor3f {
    float    r;
    float    g;
    float    b;
} XrColor3f;
```

### Member Descriptions

- **r** is the red component of the color.
- **g** is the green component of the color.
- **b** is the blue component of the color.

Unless otherwise specified, colors are encoded as linear (not with sRGB nor other gamma compression) values with individual components being in the range of 0.0 through 1.0.

The [XrColor4f](#) structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrColor4f {
    float    r;
    float    g;
    float    b;
    float    a;
} XrColor4f;
```

## Member Descriptions

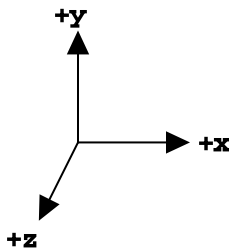
- **r** is the red component of the color.
- **g** is the green component of the color.
- **b** is the blue component of the color.
- **a** is the alpha component of the color.

Unless otherwise specified, colors are encoded as linear (not with sRGB nor other gamma compression) values with individual components being in the range of 0.0 through 1.0, and without the RGB components being premultiplied by the alpha component.

If color encoding is specified as being premultiplied by the alpha component, the RGB components are set to zero if the alpha component is zero.

## 2.18. Coordinate System

This API uses a Cartesian right-handed coordinate system.



*Figure 1. Right Handed Coordinate System*

The conventions for mapping coordinate axes of any particular space to meaningful directions depend on and are documented with the description of the space.

The API uses 2D, 3D, and 4D floating-point vectors to describe points and directions in a space.

A two-dimensional vector is defined by the [XrVector2f](#) structure:

```
typedef struct XrVector2f {  
    float    x;  
    float    y;  
} XrVector2f;
```

### Member Descriptions

- **x** is the x coordinate of the vector.
- **y** is the y coordinate of the vector.

If used to represent physical distances (rather than e.g. normalized direction) and not otherwise specified, values **must** be in meters.

A three-dimensional vector is defined by the [XrVector3f](#) structure:

```
typedef struct XrVector3f {  
    float    x;  
    float    y;  
    float    z;  
} XrVector3f;
```

### Member Descriptions

- **x** is the x coordinate of the vector.
- **y** is the y coordinate of the vector.
- **z** is the z coordinate of the vector.

If used to represent physical distances (rather than e.g. velocity or angular velocity) and not otherwise specified, values **must** be in meters.

A four-dimensional or homogeneous vector is defined by the [XrVector4f](#) structure:



```
// Provided by XR_VERSION_1_0
typedef struct XrVector4f {
    float    x;
    float    y;
    float    z;
    float    w;
} XrVector4f;
```

### Member Descriptions

- **x** is the x coordinate of the vector.
- **y** is the y coordinate of the vector.
- **z** is the z coordinate of the vector.
- **w** is the w coordinate of the vector.

If used to represent physical distances, **x**, **y**, and **z** values **must** be in meters.

Rotation is represented by a unit quaternion defined by the [XrQuaternionf](#) structure:

```
typedef struct XrQuaternionf {
    float    x;
    float    y;
    float    z;
    float    w;
} XrQuaternionf;
```

### Member Descriptions

- **x** is the x coordinate of the quaternion.
- **y** is the y coordinate of the quaternion.
- **z** is the z coordinate of the quaternion.
- **w** is the w coordinate of the quaternion.

A pose is defined by the [XrPosef](#) structure:

```
typedef struct XrPosef {
    XrQuaternionf    orientation;
    XrVector3f       position;
} XrPosef;
```

## Member Descriptions

- `orientation` is an `XrQuaternionf` representing the orientation within a space.
- `position` is an `XrVector3f` representing position within a space.

A construct representing a position and orientation within a space, with position expressed in meters, and orientation represented as a unit quaternion. When using `XrPosef` the rotation described by `orientation` is always applied before the translation described by `position`.

A runtime **must** return `XR_ERROR_POSE_INVALID` if the `orientation` norm deviates by more than 1% from unit length.

## 2.19. Common Data Types

Some OpenXR data types are used in multiple structures. Those include the `XrVector*f` family of types, the spatial types specified above, and the following categories of structures:

- offset
- extents
- rectangle
- field of view

**Offsets** are used to describe the direction and distance of an offset in two dimensions.

A floating-point offset is defined by the structure:

```
// Provided by XR_VERSION_1_0
typedef struct XrOffset2Df {
    float    x;
    float    y;
} XrOffset2Df;
```

## Member Descriptions

- `x` is the floating-point offset in the x direction.
- `y` is the floating-point offset in the y direction.

This structure is used for component values that may be real numbers, represented with single-precision floating point. For representing offsets in discrete values, such as texels, the integer variant [XrOffset2Di](#) is used instead.

If used to represent physical distances, values **must** be in meters.

An integer offset is defined by the structure:

```
typedef struct XrOffset2Di {  
    int32_t    x;  
    int32_t    y;  
} XrOffset2Di;
```

## Member Descriptions

- `x` is the integer offset in the x direction.
- `y` is the integer offset in the y direction.

This variant is for representing discrete values such as texels. For representing physical distances, the floating-point variant [XrOffset2Df](#) is used instead.

**Extents** are used to describe the size of a rectangular region in two or three dimensions.

A two-dimensional floating-point extent is defined by the structure:

```
// Provided by XR_VERSION_1_0  
typedef struct XrExtent2Df {  
    float    width;  
    float    height;  
} XrExtent2Df;
```

## Member Descriptions

- **width** is the floating-point width of the extent.
- **height** is the floating-point height of the extent.

This structure is used for component values that may be real numbers, represented with single-precision floating point. For representing extents in discrete values, such as texels, the integer variant [XrExtent2Di](#) is used instead.

If used to represent physical distances, values **must** be in meters.

The **width** and **height** value **must** be non-negative.

The [XrExtent3Df](#) structure is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrExtent3Df {
    float    width;
    float    height;
    float    depth;
} XrExtent3Df;
```

## Member Descriptions

- **width** is the floating-point width of the extent (x).
- **height** is the floating-point height of the extent (y).
- **depth** is the floating-point depth of the extent (z).

This structure is used for component values that may be real numbers, represented with single-precision floating point.

If used to represent physical distances, values **must** be in meters. The width, height, and depth values **must** be non-negative.

A two-dimensional integer extent is defined by the structure:

```
typedef struct XrExtent2Di {
    int32_t    width;
    int32_t    height;
} XrExtent2Di;
```

## Member Descriptions

- `width` is the integer width of the extent.
- `height` is the integer height of the extent.

This variant is for representing discrete values such as texels. For representing physical distances, the floating-point variant [XrExtent2Df](#) is used instead.

The `width` and `height` value **must** be non-negative.

**Rectangles** are used to describe a specific rectangular region in two dimensions. Rectangles **must** include both an offset and an extent defined in the same units. For instance, if a rectangle is in meters, both offset and extent **must** be in meters.

A rectangle with floating-point values is defined by the structure:

```
// Provided by XR_VERSION_1_0
typedef struct XrRect2Df {
    XrOffset2Df    offset;
    XrExtent2Df    extent;
} XrRect2Df;
```

## Member Descriptions

- `offset` is the [XrOffset2Df](#) specifying the rectangle offset.
- `extent` is the [XrExtent2Df](#) specifying the rectangle extent.

This structure is used for component values that may be real numbers, represented with single-precision floating point.

The `offset` is the position of the rectangle corner with minimum value coordinates. The other three corners are computed by adding the `XrExtent2Df::width` to the `x` offset, `XrExtent2Df::height` to the `y` offset, or both.

A rectangle with integer values is defined by the structure:

```
typedef struct XrRect2Di {
    XrOffset2Di    offset;
    XrExtent2Di   extent;
} XrRect2Di;
```

### Member Descriptions

- **offset** is the [XrOffset2Di](#) specifying the integer rectangle offset.
- **extent** is the [XrExtent2Di](#) specifying the integer rectangle extent.

This variant is for representing discrete values such as texels. For representing physical distances, the floating-point variant [XrRect2Df](#) is used instead.

The **offset** is the position of the rectangle corner with minimum value coordinates. The other three corners are computed by adding the [XrExtent2Di::width](#) to the **x** offset, [XrExtent2Di::height](#) to the **y** offset, or both.

An [XrSpheref](#) structure describes the center and radius of a sphere bounds.

```
// Provided by XR_VERSION_1_1
typedef struct XrSpheref {
    XrPosef    center;
    float     radius;
} XrSpheref;
```

### Member Descriptions

- **center** is an [XrPosef](#) representing the pose of the center of the sphere within the reference frame of the corresponding [XrSpace](#).
- **radius** is the finite non-negative radius of the sphere.

The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if **radius** is not a finite positive value.

An [XrBoxf](#) structure describes the pose and extents of an oriented box.

```
// Provided by XR_VERSION_1_1
typedef struct XrBoxf {
    XrPosef      center;
    XrExtent3Df  extents;
} XrBoxf;
```

## Member Descriptions

- **center** is an [XrPosef](#) defining the center position and orientation of the oriented bounding box bound within the reference frame of the corresponding [XrSpace](#).
- **extents** is an [XrExtent3Df](#) defining the edge-to-edge length of the box along each dimension with **center** as the center.

The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if width, height or depth values are negative.

An [XrFrustumf](#) structure describes the pose, field of view, and far distance of a frustum.

```
// Provided by XR_VERSION_1_1
typedef struct XrFrustumf {
    XrPosef      pose;
    XrFovf       fov;
    float        nearZ;
    float        farZ;
} XrFrustumf;
```

## Member Descriptions

- **pose** is an [XrPosef](#) defining the position and orientation of the tip of the frustum within the reference frame of the corresponding [XrSpace](#).
- **fov** is an [XrFovf](#) for the four sides of the frustum where **angleLeft** and **angleRight** are along the X axis and **angleUp** and **angleDown** are along the Y axis of the frustum space.
- **nearZ** is the positive distance of the near plane of the frustum bound along the -Z direction of the frustum space.
- **farZ** is the positive distance of the far plane of the frustum bound along the -Z direction of the frustum space.

The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if **farZ** is less than or equal to zero.

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `nearZ` is less than zero.

See [XrFovf](#) for validity requirements on `fov`.

The [XrUuid](#) structure is a 128-bit Universally Unique Identifier and is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrUuid {
    uint8_t    data[XR_UUID_SIZE];
} XrUuid;
```

### Member Descriptions

- `data` is a 128-bit Universally Unique Identifier.

The structure is composed of 16 octets, with the size and order of the fields defined in [RFC 4122 section 4.1.2](#).

## 2.20. Angles

Where a value is provided as a function parameter or as a structure member and will be interpreted as an angle, the value is defined to be in radians.

Field of view (FoV) is defined by the structure:

```
typedef struct XrFovf {
    float    angleLeft;
    float    angleRight;
    float    angleUp;
    float    angleDown;
} XrFovf;
```



## Member Descriptions

- `angleLeft` is the angle of the left side of the field of view. For a symmetric field of view this value is negative.
- `angleRight` is the angle of the right side of the field of view.
- `angleUp` is the angle of the top part of the field of view.
- `angleDown` is the angle of the bottom part of the field of view. For a symmetric field of view this value is negative.

Angles to the right of the center and upwards from the center are positive, and angles to the left of the center and down from the center are negative. The total horizontal field of view is `angleRight` minus `angleLeft`, and the total vertical field of view is `angleUp` minus `angleDown`. For a symmetric FoV, `angleRight` and `angleUp` will have positive values, `angleLeft` will be `-angleRight`, and `angleDown` will be `-angleUp`.

The angles **must** be specified in radians, and **must** be between  $-\pi/2$  and  $\pi/2$  exclusively.

When `angleLeft` > `angleRight`, the content of the view **must** be flipped horizontally. When `angleDown` > `angleUp`, the content of the view **must** be flipped vertically.

## 2.21. Boolean Values

```
typedef uint32_t XrBool32;
```

Boolean values used by OpenXR are of type `XrBool32` and are 32-bits wide as suggested by the name. The only valid values are the following:

### Enumerant Descriptions

- `XR_TRUE` represents a true value.
- `XR_FALSE` represents a false value.

```
#define XR_TRUE 1
```

```
#define XR_FALSE
```

```
0
```

## 2.22. Events

Events are messages sent from the runtime to the application.

### 2.22.1. Event Polling

Events are placed in a queue within the runtime. The application **must** read from the queue with regularity. Events are read from the queue one at a time via [xrPollEvent](#). Every type of event is identified by an individual structure type, with each such structure beginning with an [XrEventDataBaseHeader](#).

*Example 1. Proper Method for Receiving OpenXR Event Data*

```
XrInstance instance; // previously initialized

// Initialize an event buffer to hold the output.
XrEventDataBuffer event = {XR_TYPE_EVENT_DATA_BUFFER};
XrResult result = xrPollEvent(instance, &event);
if (result == XR_SUCCESS) {
    switch (event.type) {
        case XR_TYPE_EVENT_DATA_SESSION_STATE_CHANGED: {
            const XrEventDataSessionStateChanged& session_state_changed_event =
                *reinterpret_cast<XrEventDataSessionStateChanged*>(&event);
            // ...
            break;
        }
        case XR_TYPE_EVENT_DATA_INSTANCE_LOSS_PENDING: {
            const XrEventDataInstanceLossPending& instance_loss_pending_event =
                *reinterpret_cast<XrEventDataInstanceLossPending*>(&event);
            // ...
            break;
        }
    }
}
```

### **xrPollEvent**

The [xrPollEvent](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrPollEvent(
    XrInstance                instance,
    XrEventDataBuffer*       eventData);
```

## Parameter Descriptions

- `instance` is a valid `XrInstance`.
- `eventData` is a pointer to a valid `XrEventDataBuffer`.

`xrPollEvent` polls for the next event and returns an event if one is available. `xrPollEvent` returns immediately regardless of whether an event was available. The event (if present) is unilaterally removed from the queue if a valid `XrInstance` is provided. On return, the `eventData` parameter is filled with the event's data and the type field is changed to the event's type. Runtimes **may** create valid `next` chains depending on enabled extensions, but they **must** guarantee that any such chains point only to objects which fit completely within the original `XrEventDataBuffer` pointed to by `eventData`.

The runtime **must** discard queued events which contain destroyed or otherwise invalid handles. The runtime **must** not return events containing handles that have been destroyed or are otherwise invalid at the time of the call to `xrPollEvent`.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `eventData` **must** be a pointer to an `XrEventDataBuffer` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_EVENT_UNAVAILABLE`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`

Table 2. Event Descriptions

| Event  | Description   |
|--|---|
| <a href="#">XrEventDataEventsLost</a>                  | event queue has overflowed and some events were lost                                  |
| <a href="#">XrEventDataInstanceLossPending</a>         | application is about to lose the instance   |
| <a href="#">XrEventDataInteractionProfileChanged</a>   | current interaction profile for one or more top level user paths has changed          |
| <a href="#">XrEventDataReferenceSpaceChangePending</a> | runtime will begin operating with updated definitions or bounds for a reference space |
| <a href="#">XrEventDataSessionStateChanged</a>         | the application's session has changed lifecycle state                                 |

The [XrEventDataBaseHeader](#) structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrEventDataBaseHeader {
    XrStructureType    type;
    const void*       next;
} XrEventDataBaseHeader;
```

### Parameter Descriptions

- `type` is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

The [XrEventDataBaseHeader](#) is a generic structure used to identify the common event data elements.

Upon receipt, the [XrEventDataBaseHeader](#) pointer **should** be type-cast to a pointer of the appropriate event data type based on the `type` parameter.

## Valid Usage (Implicit)

- **type** **must** be one of the following `XrStructureType` values:  
`XR_TYPE_EVENT_DATA_DISPLAY_REFRESH_RATE_CHANGED_FB`, `XR_TYPE_EVENT_DATA_EVENTS_LOST`,  
`XR_TYPE_EVENT_DATA_EYE_CALIBRATION_CHANGED_ML`, `XR_TYPE_EVENT_DATA_HEADSET_FIT_CHANGED_ML`,  
`XR_TYPE_EVENT_DATA_INSTANCE_LOSS_PENDING`, `XR_TYPE_EVENT_DATA_INTERACTION_PROFILE_CHANGED`,  
`XR_TYPE_EVENT_DATA_LOCALIZATION_CHANGED_ML`,  
`XR_TYPE_EVENT_DATA_MAIN_SESSION_VISIBILITY_CHANGED_EXTX`,  
`XR_TYPE_EVENT_DATA_MARKER_TRACKING_UPDATE_VARJO`, `XR_TYPE_EVENT_DATA_PERF_SETTINGS_EXT`,  
`XR_TYPE_EVENT_DATA_REFERENCE_SPACE_CHANGE_PENDING`,  
`XR_TYPE_EVENT_DATA_SESSION_STATE_CHANGED`, `XR_TYPE_EVENT_DATA_SPACE_ERASE_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_SPACE_LIST_SAVE_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_SPACE_QUERY_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_SPACE_QUERY_RESULTS_AVAILABLE_FB`,  
`XR_TYPE_EVENT_DATA_SPACE_SAVE_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_SPACE_SET_STATUS_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_SPACE_SHARE_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_SPATIAL_ANCHOR_CREATE_COMPLETE_FB`,  
`XR_TYPE_EVENT_DATA_VISIBILITY_MASK_CHANGED_KHR`,  
`XR_TYPE_EVENT_DATA_VIVE_TRACKER_CONNECTED_HTCX`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

```
typedef struct XrEventDataBuffer {  
    XrStructureType    type;  
    const void*       next;  
    uint8_t           varying[4000];  
} XrEventDataBuffer;
```

## Parameter Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- **varying** is a fixed sized output buffer big enough to hold returned data elements for all specified event data types.

The [XrEventDataBuffer](#) is a structure passed to [xrPollEvent](#) large enough to contain any returned event data element. The maximum size is specified by [XR\\_MAX\\_EVENT\\_DATA\\_SIZE](#).

An application **can** set (or reset) only the **type** member and clear the **next** member of an `XrEventDataBuffer` before passing it as an input to `xrPollEvent`. The runtime **must** ignore the contents of the **varying** field and overwrite it without reading it.

A pointer to an `XrEventDataBuffer` **may** be type-cast to an `XrEventDataBaseHeader` pointer, or a pointer to any other appropriate event data based on the **type** parameter.

### Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_EVENT_DATA_BUFFER`
- **next** **must** be `NULL` or a valid pointer to the **next structure in a structure chain**

```
// Provided by XR_VERSION_1_0
#define XR_MAX_EVENT_DATA_SIZE sizeof(XrEventDataBuffer)
```

`XR_MAX_EVENT_DATA_SIZE` is the size of `XrEventDataBuffer`, including the size of the `XrEventDataBuffer::type` and `XrEventDataBuffer::next` members.

### XrEventDataEventsLost

The `XrEventDataEventsLost` structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrEventDataEventsLost {
    XrStructureType    type;
    const void*        next;
    uint32_t           lostEventCount;
} XrEventDataEventsLost;
```

### Member Descriptions

- **type** is the `XrStructureType` of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- **lostEventCount** is the number of events which have overflowed since the last call to `xrPollEvent`.

Receiving the [XrEventDataEventsLost](#) event structure indicates that the event queue overflowed and some events were removed at the position within the queue at which this event was found.

### Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_EVENT_DATA_EVENTS_LOST`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

Other event structures are defined in later chapters in the context where their definition is most relevant.

## 2.23. System resource lifetime

The creator of an underlying system resource is responsible for ensuring the resource's lifetime matches the lifetime of the associated OpenXR handle.

Resources passed as inputs from the application to the runtime when creating an OpenXR handle **should** not be freed while that handle is valid. A runtime **must** not free resources passed as inputs or decrease their reference counts (if applicable) from the initial value. For example, the graphics device handle (or pointer) passed in to [xrCreateSession](#) in [XrGraphicsBinding\\*](#) structure **should** be kept alive when the corresponding [XrSession](#) handle is valid, and **should** be freed by the application after the [XrSession](#) handle is destroyed.

Resources created by the runtime should not be freed by the application, and the application **should** maintain the same reference count (if applicable) at the destruction of the OpenXR handle as it had at its creation. For example, the [ID3D\\*Texture2D](#) objects in the [XrSwapchainImageD3D\\*](#) are created by the runtime and associated with the lifetime of the [XrSwapchain](#) handle. The application **should** not keep additional reference counts on any [ID3D\\*Texture2D](#) objects past the lifetime of the [XrSwapchain](#) handle, or make extra reference count decrease after destroying the [XrSwapchain](#) handle.

# Chapter 3. API Initialization

Before using an OpenXR runtime, an application **must** initialize it by creating an [XrInstance](#) object. The following functions are useful for gathering information about the API layers and extensions installed on the system and creating the instance.

## Instance Creation Functions

- [xrEnumerateApiLayerProperties](#)
- [xrEnumerateInstanceExtensionProperties](#)
- [xrCreateInstance](#)

[xrEnumerateApiLayerProperties](#) and [xrEnumerateInstanceExtensionProperties](#) **can** be called before calling [xrCreateInstance](#).

## 3.1. Exported Functions

A dynamically linked library (.dll or .so) that implements the API loader **must** export all core OpenXR API functions. However, the application **can** gain access to extension functions by obtaining pointers to these functions through the use of [xrGetInstanceProcAddr](#).

## 3.2. Function Pointers

Function pointers for all OpenXR functions **can** be obtained with the function [xrGetInstanceProcAddr](#).

```
// Provided by XR_VERSION_1_0
XrResult xrGetInstanceProcAddr(
    XrInstance          instance,
    const char*         name,
    PFN_xrVoidFunction* function);
```

## Parameter Descriptions

- **instance** is the instance that the function pointer will be compatible with, or **NULL** for functions not dependent on any instance.
- **name** is the name of the function to obtain.
- **function** is the address of the function pointer to get.



`xrGetInstanceProcAddr` itself is obtained in a platform- and loader- specific manner. Typically, the loader library will export this function as a function symbol, so applications **can** link against the loader library, or load it dynamically and look up the symbol using platform-specific APIs. Loaders **must** export function symbols for all core OpenXR functions. Because of this, applications that use only the core OpenXR functions have no need to use `xrGetInstanceProcAddr`.

Because an application **can** call `xrGetInstanceProcAddr` before creating an instance, `xrGetInstanceProcAddr` returns a valid function pointer when the `instance` parameter is `XR_NULL_HANDLE` and the `name` parameter is one of the following strings:

### No Instance Required

- `xrEnumerateInstanceExtensionProperties`
- `xrEnumerateApiLayerProperties`
- `xrCreateInstance`

`xrGetInstanceProcAddr` **must** return `XR_ERROR_HANDLE_INVALID` if `name` is not one of the above strings and `instance` is `XR_NULL_HANDLE`. `xrGetInstanceProcAddr` **may** return `XR_ERROR_HANDLE_INVALID` if `name` is not one of the above strings and `instance` is invalid but not `XR_NULL_HANDLE`.

`xrGetInstanceProcAddr` **must** return `XR_ERROR_FUNCTION_UNSUPPORTED` if `instance` is a valid instance and the string specified in `name` is not the name of an OpenXR core or enabled extension function.

If `name` is the name of an extension function, then the result returned by `xrGetInstanceProcAddr` will depend upon how the `instance` was created. If `instance` was created with the related extension’s name appearing in the `XrInstanceCreateInfo::enabledExtensionNames` array, then `xrGetInstanceProcAddr` returns a valid function pointer. If the related extension’s name did not appear in the `XrInstanceCreateInfo::enabledExtensionNames` array during the creation of `instance`, then `xrGetInstanceProcAddr` returns `XR_ERROR_FUNCTION_UNSUPPORTED`. Because of this, function pointers returned by `xrGetInstanceProcAddr` using one `XrInstance` may not be valid when used with objects related to a different `XrInstance`.

The returned function pointer is of type `PFN_xrVoidFunction`, and must be cast to the type of the function being queried.

The table below defines the various use cases for `xrGetInstanceProcAddr` and return value (“fp” is “function pointer”) for each case.

Table 3. `xrGetInstanceProcAddr` behavior

| <code>instance</code> parameter | <code>name</code> parameter | return value |
|---------------------------------|-----------------------------|--------------|
| *                               | <code>NULL</code>           | undefined    |
| invalid instance                | *                           | undefined    |

| instance parameter | name parameter   | return value    |
|--------------------|--|-----------------|
| NULL               | <a href="#">xrEnumerateInstanceExtensionProperties</a> | fp              |
| NULL               | <a href="#">xrEnumerateApiLayerProperties</a>          | fp              |
| NULL               | <a href="#">xrCreateInstance</a>                       | fp              |
| NULL               | * (any name not covered above)                         | NULL            |
| instance           | core OpenXR function                                   | fp <sup>1</sup> |
| instance           | enabled extension function for instance                | fp <sup>1</sup> |
| instance           | * (any name not covered above)                         | NULL            |

**1**  
The returned function pointer **must** only be called with a handle (the first parameter) that is `instance` or a child of `instance`.

### Valid Usage (Implicit)

- If `instance` is not `XR_NULL_HANDLE`, `instance` **must** be a valid `XrInstance` handle
- `name` **must** be a null-terminated UTF-8 string
- `function` **must** be a pointer to a `PFN_xrVoidFunction` value

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

```
typedef void (XRAPI_PTR *PFN_xrVoidFunction)(void);
```

### Parameter Descriptions

- no parameters.

[PFN\\_xrVoidFunction](#) is a generic function pointer type returned by queries, specifically those to [xrGetInstanceProcAddr](#).

```
typedef XrResult (XRAPI_PTR *PFN_xrGetInstanceProcAddr)(XrInstance instance, const char* name, PFN_xrVoidFunction* function);
```

[PFN\\_xrGetInstanceProcAddr](#) is a function pointer type for [xrGetInstanceProcAddr](#).

```
typedef struct XrApiLayerCreateInfo XrApiLayerCreateInfo;  
typedef XrResult (XRAPI_PTR *PFN_xrCreateApiLayerInstance)(  
    const XrInstanceCreateInfo* info,  
    const XrApiLayerCreateInfo* apiLayerInfo,  
    XrInstance* instance);
```

[PFN\\_xrCreateApiLayerInstance](#) is a function pointer type for [xrCreateApiLayerInstance](#).

Note: This function pointer type is only used by an OpenXR loader library, and never by an application.

## 3.3. Runtime Interface Negotiation

In order to negotiate the runtime interface version with the loader, the runtime **must** implement the [xrNegotiateLoaderRuntimeInterface](#) function.

### Note



The API described in this section is solely intended for use between an OpenXR loader and a runtime (and/or an API layer, where noted). Applications use the appropriate loader library for their platform to load the active runtime and configured API layers, rather than making these calls directly. This section is included in the specification to ensure consistency between runtimes in their interactions with the loader.

Be advised that as this is not application-facing API, some of the typical OpenXR API conventions are not followed in this section.

The `xrNegotiateLoaderRuntimeInterface` function is defined as:

```
// Provided by XR_LOADER_VERSION_1_0
XrResult xrNegotiateLoaderRuntimeInterface(
    const XrNegotiateLoaderInfo*      loaderInfo,
    XrNegotiateRuntimeRequest*       runtimeRequest);
```

### Parameter Descriptions

- `loaderInfo` **must** be a pointer to a valid `XrNegotiateLoaderInfo` structure.
- `runtimeRequest` **must** be a valid pointer to an `XrNegotiateRuntimeRequest` structure, with minimal initialization, as subsequently described, to be fully populated by the called runtime.

`xrNegotiateLoaderRuntimeInterface` **should** be directly exported by a runtime so that using e.g. `GetProcAddress` on Windows or `dlsym` on POSIX platforms returns a valid function pointer to it.

The runtime **must** return `XR_ERROR_INITIALIZATION_FAILED` if any of the following conditions on `loaderInfo` are true:

- `XrNegotiateLoaderInfo::structType` is not `XR_LOADER_INTERFACE_STRUCT_LOADER_INFO`
- `XrNegotiateLoaderInfo::structVersion` is not `XR_LOADER_INFO_STRUCT_VERSION`
- `XrNegotiateLoaderInfo::structSize` is not `sizeof(XrNegotiateLoaderInfo)`

The runtime **must** also return `XR_ERROR_INITIALIZATION_FAILED` if any of the following conditions on `runtimeRequest` are true:

- `XrNegotiateRuntimeRequest::structType` is not `XR_LOADER_INTERFACE_STRUCT_RUNTIME_REQUEST`
- `XrNegotiateRuntimeRequest::structVersion` is not `XR_RUNTIME_INFO_STRUCT_VERSION`
- `XrNegotiateRuntimeRequest::structSize` is not `sizeof(XrNegotiateRuntimeRequest)`

The runtime **must** determine if it supports the loader's request. The runtime does not support the loader's request if either of the following is true:

- the runtime does not support the interface versions supported by the loader as specified by the parameters `XrNegotiateLoaderInfo::minInterfaceVersion` and `XrNegotiateLoaderInfo::maxInterfaceVersion`
- the runtime does not support the API versions supported by the loader as specified by the parameters `XrNegotiateLoaderInfo::minApiVersion` and `XrNegotiateLoaderInfo::maxApiVersion`.

The runtime **must** return `XR_ERROR_INITIALIZATION_FAILED` if it does not support the loader's request.

If the function succeeds, the runtime **must** set the `XrNegotiateRuntimeRequest::runtimeInterfaceVersion` with the runtime interface version it desires to support. The `XrNegotiateRuntimeRequest::runtimeInterfaceVersion` set **must** be between `XrNegotiateLoaderInfo::minInterfaceVersion` and `XrNegotiateLoaderInfo::maxInterfaceVersion`.

If the function succeeds, the runtime **must** set the `XrNegotiateRuntimeRequest::runtimeApiVersion` with the API version of OpenXR it will execute under. The `XrNegotiateRuntimeRequest::runtimeApiVersion` set **must** be between `XrNegotiateLoaderInfo::minApiVersion` and `XrNegotiateLoaderInfo::maxApiVersion`.

If the function succeeds, the runtime **must** set the `XrNegotiateRuntimeRequest::getInstanceProcAddr` with a valid function pointer for the loader to use to query function pointers to the remaining OpenXR functions supported by the runtime.

If the function succeeds, the runtime **must** return `XR_SUCCESS`.

### Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to calling `xrNegotiateLoaderRuntimeInterface`
- `loaderInfo` **must** be a pointer to a valid `XrNegotiateLoaderInfo` structure
- `runtimeRequest` **must** be a pointer to an `XrNegotiateRuntimeRequest` structure

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_INITIALIZATION_FAILED`

The `XrNegotiateLoaderInfo` structure is used to pass information about the loader to a runtime or an API layer.

The [XrNegotiateLoaderInfo](#) structure is defined as:

```
typedef struct XrNegotiateLoaderInfo {
    XrLoaderInterfaceStructs    structType;
    uint32_t                    structVersion;
    size_t                      structSize;
    uint32_t                    minInterfaceVersion;
    uint32_t                    maxInterfaceVersion;
    XrVersion                   minApiVersion;
    XrVersion                   maxApiVersion;
} XrNegotiateLoaderInfo;
```

## Member Descriptions

- `structType` **must** be [XR\\_LOADER\\_INTERFACE\\_STRUCT\\_LOADER\\_INFO](#).
- `structVersion` **must** be a valid version of the structure. The value [XR\\_LOADER\\_INFO\\_STRUCT\\_VERSION](#) describes the current latest version of this structure.
- `structSize` **must** be the size in bytes of the current version of the structure (i.e. `sizeof(XrNegotiateLoaderInfo)`).
- `minInterfaceVersion` is the minimum runtime or API layer interface version supported by the loader.
- `maxInterfaceVersion` is the maximum valid version of the runtime or API layer interface version supported by the loader, currently defined using [XR\\_CURRENT\\_LOADER\\_RUNTIME\\_VERSION](#) or [XR\\_CURRENT\\_LOADER\\_API\\_LAYER\\_VERSION](#).
- `minApiVersion` is the minimum supported version of the OpenXR API by the loader as formatted by [XR\\_MAKE\\_VERSION](#). Patch is ignored.
- `maxApiVersion` is the maximum supported version of the OpenXR API by the loader as formatted by [XR\\_MAKE\\_VERSION](#). Patch is ignored.

This structure is an input from the loader to the runtime in an [xrNegotiateLoaderRuntimeInterface](#) call, as well as from the loader to an API layer in an [xrNegotiateLoaderApiLayerInterface](#) call.

## Valid Usage (Implicit)

- The [XR\\_LOADER\\_VERSION\\_1\\_0](#) extension **must** be enabled prior to using [XrNegotiateLoaderInfo](#)
- `structType` **must** be a valid [XrLoaderInterfaceStructs](#) value

The [XrLoaderInterfaceStructs](#) enumeration is defined as:

```
typedef enum XrLoaderInterfaceStructs {
    XR_LOADER_INTERFACE_STRUCT_UNINITIALIZED = 0,
    XR_LOADER_INTERFACE_STRUCT_LOADER_INFO = 1,
    XR_LOADER_INTERFACE_STRUCT_API_LAYER_REQUEST = 2,
    XR_LOADER_INTERFACE_STRUCT_RUNTIME_REQUEST = 3,
    XR_LOADER_INTERFACE_STRUCT_API_LAYER_CREATE_INFO = 4,
    XR_LOADER_INTERFACE_STRUCT_API_LAYER_NEXT_INFO = 5,
    XR_LOADER_INTERFACE_STRUCTS_MAX_ENUM = 0x7FFFFFFF
} XrLoaderInterfaceStructs;
```

This enumeration serves a similar purpose in the runtime and API layer interface negotiation (loader) API as [XrStructureType](#) serves in the application-facing API.

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_LOADER_INFO_STRUCT_VERSION 1
```

[XR\\_LOADER\\_INFO\\_STRUCT\\_VERSION](#) is the current version of the [XrNegotiateLoaderInfo](#) structure. It is used to populate the [XrNegotiateLoaderInfo::structVersion](#) field.

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_CURRENT_LOADER_RUNTIME_VERSION 1
```

[XR\\_CURRENT\\_LOADER\\_RUNTIME\\_VERSION](#) is the current version of the overall OpenXR Loader Runtime interface. It is used to populate maximum and minimum interface version fields in [XrNegotiateLoaderInfo](#) when loading a runtime.

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_CURRENT_LOADER_API_LAYER_VERSION 1
```

[XR\\_CURRENT\\_LOADER\\_API\\_LAYER\\_VERSION](#) is the current version of the overall OpenXR Loader API Layer interface. It is used to populate maximum and minimum interface version fields in [XrNegotiateLoaderInfo](#) when loading an API layer.

The [XrNegotiateRuntimeRequest](#) structure is used to pass information about the runtime back to the loader.

The `XrNegotiateRuntimeRequest` structure is defined as:

```
typedef struct XrNegotiateRuntimeRequest {
    XrLoaderInterfaceStructs    structType;
    uint32_t                    structVersion;
    size_t                      structSize;
    uint32_t                    runtimeInterfaceVersion;
    XrVersion                   runtimeApiVersion;
    PFN_xrGetInstanceProcAddr  getInstanceProcAddr;
} XrNegotiateRuntimeRequest;
```

### Member Descriptions

- `structType` **must** be `XR_LOADER_INTERFACE_STRUCT_RUNTIME_REQUEST`.
- `structVersion` **must** be a valid version of the structure. The value `XR_RUNTIME_INFO_STRUCT_VERSION` is used to describe the current version of this structure.
- `structSize` **must** be the size in bytes of the current version of the structure (i.e. `sizeof(XrNegotiateRuntimeRequest)`)
- `runtimeInterfaceVersion` is the version of the runtime interface version being requested by the runtime. Must: not be outside of the bounds of the `XrNegotiateLoaderInfo::minInterfaceVersion` and `XrNegotiateLoaderInfo::maxInterfaceVersion` values (inclusive).
- `runtimeApiVersion` is the version of the OpenXR API supported by this runtime as formatted by `XR_MAKE_VERSION`. Patch is ignored.
- `getInstanceProcAddr` is a pointer to the runtime's `xrGetInstanceProcAddr` implementation that will be used by the loader to populate a dispatch table of OpenXR functions supported by the runtime.

This is an output structure from runtime negotiation. The loader **must** populate `structType`, `structVersion`, and `structSize` to ensure correct interpretation by the runtime, while the runtime populates the rest of the fields in a successful call to `xrNegotiateLoaderRuntimeInterface`.



## Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to using `XrNegotiateRuntimeRequest`
- `structType` **must** be a valid `XrLoaderInterfaceStructs` value
- `getInstanceProcAddr` **must** be a valid `PFN_xrGetInstanceProcAddr` value

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_RUNTIME_INFO_STRUCT_VERSION 1
```

`XR_RUNTIME_INFO_STRUCT_VERSION` is the current version of the `XrNegotiateRuntimeRequest` structure. It is used to populate the `XrNegotiateRuntimeRequest::structVersion` field.

## 3.4. API Layer Interface Negotiation

In order to negotiate the API layer interface version with the loader, an OpenXR API layer **must** implement the `xrNegotiateLoaderApiLayerInterface` function.

### Note



The API described in this section is solely intended for use between an OpenXR loader and an API layer. Applications use the appropriate loader library for their platform to load the active runtime and configured API layers, rather than making these calls directly. This section is included in the specification to ensure consistency between runtimes in their interactions with the loader.

Be advised that as this is not application-facing API, some of the typical OpenXR API conventions are not followed in this section.

The `xrNegotiateLoaderApiLayerInterface` function is defined as:

```
// Provided by XR_LOADER_VERSION_1_0
XrResult xrNegotiateLoaderApiLayerInterface(
    const XrNegotiateLoaderInfo*    loaderInfo,
    const char*                      layerName,
    XrNegotiateApiLayerRequest*    apiLayerRequest);
```

## Parameter Descriptions

- `loaderInfo` **must** be a pointer to a valid `XrNegotiateLoaderInfo` structure.
- `layerName` **must** be NULL or a valid C-style NULL-terminated string listing the name of an API layer which the loader is attempting to negotiate with.
- `apiLayerRequest` **must** be a valid pointer to an `XrNegotiateApiLayerRequest` structure, with minimal initialization, as subsequently described, to be fully populated by the called API layer.

`xrNegotiateLoaderApiLayerInterface` **should** be directly exported by an API layer so that using e.g. `GetProcAddress` on Windows or `dlsym` on POSIX platforms returns a valid function pointer to it.

The API layer **must** return `XR_ERROR_INITIALIZATION_FAILED` if any of the following conditions on `loaderInfo` are true:

- `XrNegotiateLoaderInfo::structType` is not `XR_LOADER_INTERFACE_STRUCT_LOADER_INFO`
- `XrNegotiateLoaderInfo::structVersion` is not `XR_LOADER_INFO_STRUCT_VERSION`
- `XrNegotiateLoaderInfo::structSize` is not `sizeof(XrNegotiateLoaderInfo)`

The API layer **must** also return `XR_ERROR_INITIALIZATION_FAILED` if any of the following conditions on `apiLayerRequest` are true:

- `XrNegotiateApiLayerRequest::structType` is not `XR_LOADER_INTERFACE_STRUCT_API_LAYER_REQUEST`
- `XrNegotiateApiLayerRequest::structVersion` is not `XR_API_LAYER_INFO_STRUCT_VERSION`
- `XrNegotiateApiLayerRequest::structSize` is not `sizeof(XrNegotiateApiLayerRequest)`

The API layer **must** determine if it supports the loader's request. The API layer does not support the loader's request if either of the following is true:

- the API layer does not support the interface versions supported by the loader as specified by the parameters `XrNegotiateLoaderInfo::minInterfaceVersion` and `XrNegotiateLoaderInfo::maxInterfaceVersion`
- the API layer does not support the API versions supported by the loader as specified by the parameters `XrNegotiateLoaderInfo::minApiVersion` and `XrNegotiateLoaderInfo::maxApiVersion`.

The API layer **must** return `XR_ERROR_INITIALIZATION_FAILED` if it does not support the loader's request.

If the function succeeds, the API layer **must** set the `XrNegotiateApiLayerRequest::layerInterfaceVersion` with the API layer interface version it desires to support. The `XrNegotiateApiLayerRequest::layerInterfaceVersion` set **must** be between `XrNegotiateLoaderInfo::minInterfaceVersion` and `XrNegotiateLoaderInfo::maxInterfaceVersion`.

If the function succeeds, the API layer **must** set the `XrNegotiateApiLayerRequest::layerApiVersion` with

the API version of OpenXR it will execute under. The `XrNegotiateApiLayerRequest::layerApiVersion` set **must** be between `XrNegotiateLoaderInfo::minApiVersion` and `XrNegotiateLoaderInfo::maxApiVersion`.

If the function succeeds, the API layer **must** set the `XrNegotiateApiLayerRequest::getInstanceProcAddr` with a valid function pointer for the loader to use to query function pointers to the remaining OpenXR functions supported by the API layer.

If the function succeeds, the API layer **must** set the `XrNegotiateApiLayerRequest::createApiLayerInstance` with a valid function pointer to an implementation of `xrCreateApiLayerInstance` for the loader to use to create the instance through the API layer call chain.

If the function succeeds, the API layer **must** return `XR_SUCCESS`.

The API layer **must** not call into another API layer from its implementation of the `xrNegotiateLoaderApiLayerInterface` function. The loader **must** handle all API layer negotiations with each API layer individually.

### Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to calling `xrNegotiateLoaderApiLayerInterface`
- `loaderInfo` **must** be a pointer to a valid `XrNegotiateLoaderInfo` structure
- `layerName` **must** be a null-terminated UTF-8 string
- `apiLayerRequest` **must** be a pointer to an `XrNegotiateApiLayerRequest` structure

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_INITIALIZATION_FAILED`

The `XrNegotiateApiLayerRequest` structure is used to pass information about the API layer back to the loader.

The `XrNegotiateApiLayerRequest` structure is defined as:

```

typedef struct XrNegotiateApiLayerRequest {
    XrLoaderInterfaceStructs    structType;
    uint32_t                    structVersion;
    size_t                      structSize;
    uint32_t                    layerInterfaceVersion;
    XrVersion                   layerApiVersion;
    PFN_xrGetInstanceProcAddr   getInstanceProcAddr;
    PFN_xrCreateApiLayerInstance createApiLayerInstance;
} XrNegotiateApiLayerRequest;

```

## Member Descriptions

- `structType` **must** be `XR_LOADER_INTERFACE_STRUCT_API_LAYER_REQUEST`.
- `structVersion` **must** be a valid version of the structure. The value `XR_API_LAYER_INFO_STRUCT_VERSION` is used to describe the current latest version of this structure.
- `structSize` **must** be the size in bytes of the current version of the structure (i.e. `sizeof(XrNegotiateApiLayerRequest)`).
- `layerInterfaceVersion` is the version of the API layer interface version being requested by the API layer. Should not be outside of the bounds of the `XrNegotiateLoaderInfo::minInterfaceVersion` and `XrNegotiateLoaderInfo::maxInterfaceVersion` values (inclusive).
- `layerApiVersion` is the version of the OpenXR API supported by this API layer as formatted by `XR_MAKE_VERSION`. Patch is ignored.
- `getInstanceProcAddr` is a pointer to the API layer's `xrGetInstanceProcAddr` implementation that will be used by the loader to populate a dispatch table of OpenXR functions supported by the API layer.
- `createApiLayerInstance` is a pointer to the API layer's `xrCreateApiLayerInstance` implementation that will be used by the loader during a call to `xrCreateInstance` when an API layer is active. This is used because API layers need additional information at `xrCreateInstance` time.

This is an output structure from API layer negotiation. The loader **must** populate `structType`, `structVersion`, and `structSize` before calling to ensure correct interpretation by the API layer, while the API layer populates the rest of the fields in a successful call to `xrNegotiateLoaderApiLayerInterface`.

## Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to using `XrNegotiateApiLayerRequest`
- `structType` **must** be a valid `XrLoaderInterfaceStructs` value
- `getInstanceProcAddr` **must** be a valid `PFN_xrGetInstanceProcAddr` value
- `createApiLayerInstance` **must** be a valid `PFN_xrCreateApiLayerInstance` value

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_API_LAYER_INFO_STRUCT_VERSION 1
```

`XR_API_LAYER_INFO_STRUCT_VERSION` is the current version of the `XrNegotiateApiLayerRequest` structure. It is used to populate the `XrNegotiateApiLayerRequest::structVersion` field.

The `xrCreateApiLayerInstance` function is defined as:

```
// Provided by XR_LOADER_VERSION_1_0
XrResult xrCreateApiLayerInstance(
    const XrInstanceCreateInfo*      info,
    const XrApiLayerCreateInfo*     layerInfo,
    XrInstance*                      instance);
```

## Parameter Descriptions

- `info` is a pointer to the `XrInstanceCreateInfo` information passed by the application into the outer `xrCreateInstance` function.
- `layerInfo` is a pointer to an `XrApiLayerCreateInfo` structure that contains special information required by a API layer during its create instance process. This is generated by the loader.
- `instance` is a pointer to store the returned instance in, just as in the standard `xrCreateInstance` function.

An API layer's implementation of the `xrCreateApiLayerInstance` function is invoked during the loader's implementation of `xrCreateInstance`, if the layer in question is enabled.

An API layer needs additional information during `xrCreateInstance` calls, so each API layer **must** implement the `xrCreateApiLayerInstance` function, which is a special API layer function.

An API layer **must** not implement `xrCreateInstance`.

`xrCreateApiLayerInstance` **must** be called by the loader during its implementation of the `xrCreateInstance` function.

The loader **must** call the first API layer's `xrCreateApiLayerInstance` function passing in the pointer to the created `XrApiLayerCreateInfo`.

The `XrApiLayerCreateInfo::nextInfo` **must** be a linked-list of `XrApiLayerNextInfo` structures with information about each of the API layers that are to be enabled. Note that this does not operate like a `next` chain in the OpenXR application API, but instead describes the enabled API layers from outermost to innermost.

The API layer **may** validate that it is getting the correct next information by checking that the `XrApiLayerNextInfo::layerName` matches the expected value.

The API layer **must** use the information in its `XrApiLayerNextInfo` to call down the call chain to the next `xrCreateApiLayerInstance`:

- The API layer **must** copy the `XrApiLayerCreateInfo` structure into its own structure.
- The API layer **must** then update its copy of the `XrApiLayerCreateInfo` structure, setting `XrApiLayerCreateInfo::XrApiLayerCreateInfo::nextInfo` to point to the `XrApiLayerNextInfo` for the next API layer (e.g. `layerInfoCopy->nextInfo = layerInfo->nextInfo->next;`).
- The API layer **must** then use the pointer to its `XrApiLayerCreateInfo` structure (instead of the one that was passed in) when it makes a call to the `xrCreateApiLayerInstance` function.
- If the nested `xrCreateApiLayerInstance` call succeeds, the API layer **may** choose to setup its own dispatch table to the next API layer's functions using the returned `XrInstance` and the next API layer's `xrGetInstanceProcAddr`.
- The API layer **must** return the `XrResult` returned from the next API layer.

### Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to calling `xrCreateApiLayerInstance`
- `info` **must** be a pointer to a valid `XrInstanceCreateInfo` structure
- `layerInfo` **must** be a pointer to a valid `XrApiLayerCreateInfo` structure
- `instance` **must** be a pointer to an `XrInstance` handle

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_INITIALIZATION_FAILED`

The `XrApiLayerCreateInfo` structure contains special information required by a API layer during its create instance process.

The `XrApiLayerCreateInfo` structure is defined as:

```
typedef struct XrApiLayerCreateInfo {
    XrLoaderInterfaceStructs    structType;
    uint32_t                    structVersion;
    size_t                      structSize;
    void*                       loaderInstance;
    char                        settings_file_location
[XR_API_LAYER_MAX_SETTINGS_PATH_SIZE];
    XrApiLayerNextInfo*        nextInfo;
} XrApiLayerCreateInfo;
```

## Member Descriptions

- `structType` **must** be `XR_LOADER_INTERFACE_STRUCT_API_LAYER_CREATE_INFO`.
- `structVersion` is the version of the structure being supplied by the loader (i.e. `XR_API_LAYER_CREATE_INFO_STRUCT_VERSION`)
- `structSize` **must** be the size in bytes of the current version of the structure (i.e. `sizeof(XrApiLayerCreateInfo)`)
- `loaderInstance` is deprecated and **must** be ignored.
- `settings_file_location` is the location of any usable API layer settings file. The size of `settings_file_location` is given by `XR_API_LAYER_MAX_SETTINGS_PATH_SIZE`. This is currently unused.
- `nextInfo` is a pointer to the `XrApiLayerNextInfo` structure which contains information to work with the next API layer in the chain.

## Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to using `XrApiLayerCreateInfo`
- `structType` **must** be a valid `XrLoaderInterfaceStructs` value
- `loaderInstance` **must** be a pointer value
- `settings_file_location` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_API_LAYER_MAX_SETTINGS_PATH_SIZE`
- `nextInfo` **must** be a pointer to an `XrApiLayerNextInfo` structure

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_API_LAYER_CREATE_INFO_STRUCT_VERSION 1
```

`XR_API_LAYER_CREATE_INFO_STRUCT_VERSION` is the current version of the `XrApiLayerCreateInfo` structure. It is used to populate the `XrApiLayerCreateInfo::structVersion` field.

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_API_LAYER_MAX_SETTINGS_PATH_SIZE 512
```

`XR_API_LAYER_MAX_SETTINGS_PATH_SIZE` is the size of the `XrApiLayerCreateInfo::settings_file_location` field.

The `XrApiLayerNextInfo` structure:

The `XrApiLayerNextInfo` structure is defined as:

```
typedef struct XrApiLayerNextInfo {
    XrLoaderInterfaceStructs    structType;
    uint32_t                    structVersion;
    size_t                      structSize;
    char                        layerName[XR_MAX_API_LAYER_NAME_SIZE];
    PFN_xrGetInstanceProcAddr   nextGetInstanceProcAddr;
    PFN_xrCreateApiLayerInstance nextCreateApiLayerInstance;
    struct XrApiLayerNextInfo*  next;
} XrApiLayerNextInfo;
```



## Member Descriptions

- `structType` **must** be `XR_LOADER_INTERFACE_STRUCT_API_LAYER_NEXT_INFO`
- `structVersion` **must** be a valid version of the structure and the version being supplied by the loader (i.e. `XR_API_LAYER_NEXT_INFO_STRUCT_VERSION`).
- `structSize` **must** be the size in bytes of the current version of the structure (i.e. `sizeof(XrApiLayerNextInfo)`)
- `layerName` is the name of the intended next API layer, used to verify and debug the API layer chain.
- `nextGetInstanceProcAddr` is a pointer to the next API layer's `xrGetInstanceProcAddr`. This is intended for use in populating a dispatch table to the next implementations in the chain.
- `nextCreateApiLayerInstance` is a pointer to the `xrCreateApiLayerInstance` function implementation in the next API layer. This is to be called **after** the API layer has done any localized creation, but **before** the API layer records any function addresses from the next API layer using `xrGetInstanceProcAddr`.
- `next` is a pointer to the `XrApiLayerNextInfo` for the next API layer. If no API layer is after this, it will be `NULL`.

## Valid Usage (Implicit)

- The `XR_LOADER_VERSION_1_0` extension **must** be enabled prior to using `XrApiLayerNextInfo`
- `structType` **must** be a valid `XrLoaderInterfaceStructs` value
- `layerName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_API_LAYER_NAME_SIZE`
- `nextGetInstanceProcAddr` **must** be a valid `PFN_xrGetInstanceProcAddr` value
- `nextCreateApiLayerInstance` **must** be a valid `PFN_xrCreateApiLayerInstance` value
- `next` **must** be a pointer to an `XrApiLayerNextInfo` structure

```
// Provided by XR_LOADER_VERSION_1_0
#define XR_API_LAYER_NEXT_INFO_STRUCT_VERSION 1
```

`XR_API_LAYER_NEXT_INFO_STRUCT_VERSION` is the current version of the `XrApiLayerNextInfo` structure. It is used to populate the `XrApiLayerNextInfo::structVersion` field.

# Chapter 4. Instance

```
XR_DEFINE_HANDLE(XrInstance)
```

An OpenXR instance is an object that allows an OpenXR application to communicate with an OpenXR runtime. The application accomplishes this communication by calling `xrCreateInstance` and receiving a handle to the resulting `XrInstance` object.

The `XrInstance` object stores and tracks OpenXR-related application state, without storing any such state in the application's global address space. This allows the application to create multiple instances as well as safely encapsulate the application's OpenXR state since this object is opaque to the application. OpenXR runtimes **may** limit the number of simultaneous `XrInstance` objects that may be created and used, but they **must** support the creation and usage of at least one `XrInstance` object per process.

Physically, this state **may** be stored in any of the OpenXR loader, OpenXR API layers or the OpenXR runtime components. The exact storage and distribution of this saved state is implementation-dependent, except where indicated by this specification.

The tracking of OpenXR state in the instance allows the streamlining of the API, where the intended instance is inferred from the highest ascendant of an OpenXR function's target object. For example, in:

```
myResult = xrEndFrame(mySession, &myEndFrameDescription);
```

the `XrSession` object was created from an `XrInstance` object. The OpenXR loader typically keeps track of the `XrInstance` that is the parent of the `XrSession` object in this example and directs the function to the runtime associated with that instance. This tracking of OpenXR objects eliminates the need to specify an `XrInstance` in every OpenXR function.

## 4.1. API Layers and Extensions

Additional functionality **may** be provided by API layers or extensions. An API layer **must** not add or modify the definition of OpenXR functions, while an extension **may** do so.

The set of API layers to enable is specified when creating an instance, and those API layers are able to intercept any functions dispatched to that instance or any of its child objects.

Example API layers **may** include (but are not limited to):

- an API layer to dump out OpenXR API calls
- an API layer to perform OpenXR validation

To determine what set of API layers are available, OpenXR provides the [xrEnumerateApiLayerProperties](#) function:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateApiLayerProperties(
    uint32_t                propertyCapacityInput,
    uint32_t*               propertyCountOutput,
    XrApiLayerProperties*   properties);
```

### Parameter Descriptions

- `propertyCapacityInput` is the capacity of the `properties` array, or 0 to indicate a request to retrieve the required capacity.
- `propertyCountOutput` is a pointer to the count of `properties` written, or a pointer to the required capacity in the case that `propertyCapacityInput` is insufficient.
- `properties` is a pointer to an array of [XrApiLayerProperties](#) structures, but **can** be `NULL` if `propertyCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `properties` size.

The list of available layers may change at any time due to actions outside of the OpenXR runtime, so two calls to [xrEnumerateApiLayerProperties](#) with the same parameters **may** return different results, or retrieve different `propertyCountOutput` values or `properties` contents.

Once an instance has been created, the layers enabled for that instance will continue to be enabled and valid for the lifetime of that instance, even if some of them become unavailable for future instances.

### Valid Usage (Implicit)

- `propertyCountOutput` **must** be a pointer to a `uint32_t` value
- If `propertyCapacityInput` is not 0, `properties` **must** be a pointer to an array of `propertyCapacityInput` [XrApiLayerProperties](#) structures

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `XrApiLayerProperties` structure is defined as:

```
typedef struct XrApiLayerProperties {
    XrStructureType    type;
    void*              next;
    char                layerName[XR_MAX_API_LAYER_NAME_SIZE];
    XrVersion          specVersion;
    uint32_t           layerVersion;
    char               description[XR_MAX_API_LAYER_DESCRIPTION_SIZE];
} XrApiLayerProperties;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `layerName` is a string specifying the name of the API layer. Use this name in the `XrInstanceCreateInfo::enabledApiLayerNames` array to enable this API layer for an instance.
- `specVersion` is the API version the API layer was written to, encoded as described in the [API Version Numbers and Semantics](#) section.
- `layerVersion` is the version of this API layer. It is an integer, increasing with backward compatible changes.
- `description` is a string providing additional details that **can** be used by the application to identify the API layer.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_API_LAYER_PROPERTIES`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

To enable a layer, the name of the layer **should** be added to `XrInstanceCreateInfo::enabledApiLayerNames` when creating an `XrInstance`.

Loader implementations **may** provide mechanisms outside this API for enabling specific API layers. API layers enabled through such a mechanism are implicitly enabled, while API layers enabled by including the API layer name in `XrInstanceCreateInfo::enabledApiLayerNames` are explicitly enabled. Except where otherwise specified, implicitly enabled and explicitly enabled API layers differ only in the way they are enabled. Explicitly enabling an API layer that is implicitly enabled has no additional effect.

Instance extensions are able to affect the operation of the instance and any of its child objects. As stated [earlier](#), extensions can expand the OpenXR API and provide new functions or augment behavior.

Examples of extensions **may** be (but are not limited to):

## Extension Examples

- an extension to include OpenXR functions to work with a new graphics API
- an extension to expose debug information via a callback

The application can determine the available instance extensions by calling `xrEnumerateInstanceExtensionProperties`:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateInstanceExtensionProperties(
    const char*          layerName,
    uint32_t            propertyCapacityInput,
    uint32_t*           propertyCountOutput,
    XrExtensionProperties* properties);
```

## Parameter Descriptions

- `layerName` is either `NULL` or a pointer to a string naming the API layer to retrieve extensions from, as returned by `xrEnumerateApiLayerProperties`.
- `propertyCapacityInput` is the capacity of the `properties` array, or `0` to indicate a request to retrieve the required capacity.
- `propertyCountOutput` is a pointer to the count of `properties` written, or a pointer to the required capacity in the case that `propertyCapacityInput` is insufficient.
- `properties` is a pointer to an array of `XrExtensionProperties` structures, but **can** be `NULL` if `propertyCapacityInput` is `0`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `properties` size.

Because the list of available layers may change externally between calls to `xrEnumerateInstanceExtensionProperties`, two calls **may** retrieve different results if a `layerName` is available in one call but not in another. The extensions supported by a layer may also change between two calls, e.g. if the layer implementation is replaced by a different version between those calls.

## Valid Usage (Implicit)

- If `layerName` is not `NULL`, `layerName` **must** be a null-terminated UTF-8 string
- `propertyCountOutput` **must** be a pointer to a `uint32_t` value
- If `propertyCapacityInput` is not `0`, `properties` **must** be a pointer to an array of `propertyCapacityInput` `XrExtensionProperties` structures

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_RUNTIME_UNAVAILABLE`
- `XR_ERROR_API_LAYER_NOT_PRESENT`

The `XrExtensionProperties` structure is defined as:

```
typedef struct XrExtensionProperties {
    XrStructureType    type;
    void*              next;
    char                extensionName[XR_MAX_EXTENSION_NAME_SIZE];
    uint32_t            extensionVersion;
} XrExtensionProperties;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `extensionName` is a `NULL` terminated string specifying the name of the extension.
- `extensionVersion` is the version of this extension. It is an integer, incremented with backward compatible changes.

### Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_EXTENSION_PROPERTIES`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 4.2. Instance Lifecycle

The `xrCreateInstance` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrCreateInstance(
    const XrInstanceCreateInfo*    createInfo,
    XrInstance*                    instance);
```

## Parameter Descriptions

- `createInfo` points to an instance of `XrInstanceCreateInfo` controlling creation of the instance.
- `instance` points to an `XrInstance` handle in which the resulting instance is returned.

`xrCreateInstance` creates the `XrInstance`, then enables and initializes global API layers and extensions requested by the application. If an extension is provided by an API layer, both the API layer and extension **must** be specified at `xrCreateInstance` time. If a specified API layer cannot be found, no `XrInstance` will be created and the function will return `XR_ERROR_API_LAYER_NOT_PRESENT`. Likewise, if a specified extension cannot be found, the call **must** return `XR_ERROR_EXTENSION_NOT_PRESENT` and no `XrInstance` will be created. Additionally, some runtimes **may** limit the number of concurrent instances that may be in use. If the application attempts to create more instances than a runtime can simultaneously support, `xrCreateInstance` **may** return `XR_ERROR_LIMIT_REACHED`.

If the `XrApplicationInfo::applicationName` is the empty string the runtime **must** return `XR_ERROR_NAME_INVALID`.

If the `XrInstanceCreateInfo` structure contains a platform-specific extension for a platform other than the target platform, `XR_ERROR_INITIALIZATION_FAILED` **may** be returned. If a mandatory platform-specific extension is defined for the target platform but no matching extension struct is provided in `XrInstanceCreateInfo` the runtime **must** return `XR_ERROR_INITIALIZATION_FAILED`.

## Valid Usage (Implicit)

- `createInfo` **must** be a pointer to a valid `XrInstanceCreateInfo` structure
- `instance` **must** be a pointer to an `XrInstance` handle



## Return Codes

### Success

- XR\_SUCCESS

### Failure

- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_LIMIT\_REACHED
- XR\_ERROR\_RUNTIME\_UNAVAILABLE
- XR\_ERROR\_NAME\_INVALID
- XR\_ERROR\_INITIALIZATION\_FAILED
- XR\_ERROR\_EXTENSION\_NOT\_PRESENT
- XR\_ERROR\_EXTENSION\_DEPENDENCY\_NOT\_ENABLED
- XR\_ERROR\_API\_VERSION\_UNSUPPORTED
- XR\_ERROR\_API\_LAYER\_NOT\_PRESENT

The `XrInstanceCreateInfo` structure is defined as:

```
typedef struct XrInstanceCreateInfo {
    XrStructureType      type;
    const void*          next;
    XrInstanceCreateFlags createFlags;
    XrApplicationInfo    applicationInfo;
    uint32_t             enabledApiLayerCount;
    const char* const*   enabledApiLayerNames;
    uint32_t             enabledExtensionCount;
    const char* const*   enabledExtensionNames;
} XrInstanceCreateInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `createFlags` is a bitmask of [XrInstanceCreateFlags](#) that identifies options that apply to the creation.
- `applicationInfo` is an instance of [XrApplicationInfo](#). This information helps runtimes recognize behavior inherent to classes of applications. [XrApplicationInfo](#) is defined in detail below.
- `enabledApiLayerCount` is the number of global API layers to enable.
- `enabledApiLayerNames` is a pointer to an array of `enabledApiLayerCount` strings containing the names of API layers to enable for the created instance. See the [API Layers and Extensions](#) section for further details.
- `enabledExtensionCount` is the number of global extensions to enable.
- `enabledExtensionNames` is a pointer to an array of `enabledExtensionCount` strings containing the names of extensions to enable.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_INSTANCE_CREATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrDebugUtilsMessengerCreateInfoEXT](#), [XrInstanceCreateInfoAndroidKHR](#)
- `createFlags` **must** be `0`
- `applicationInfo` **must** be a valid [XrApplicationInfo](#) structure
- If `enabledApiLayerCount` is not `0`, `enabledApiLayerNames` **must** be a pointer to an array of `enabledApiLayerCount` null-terminated UTF-8 strings
- If `enabledExtensionCount` is not `0`, `enabledExtensionNames` **must** be a pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings

The [XrInstanceCreateInfo::createFlags](#) member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in [XrInstanceCreateFlagBits](#).

```
typedef XrFlags64 XrInstanceCreateFlags;
```

Valid bits for [XrInstanceCreateFlags](#) are defined by [XrInstanceCreateFlagBits](#).

```
// Flag bits for XrInstanceCreateFlags
```

There are currently no instance creation flag bits defined. This is reserved for future use.

The [XrApplicationInfo](#) structure is defined as:

```
typedef struct XrApplicationInfo {  
    char        applicationName[XR_MAX_APPLICATION_NAME_SIZE];  
    uint32_t    applicationVersion;  
    char        engineName[XR_MAX_ENGINE_NAME_SIZE];  
    uint32_t    engineVersion;  
    XrVersion   apiVersion;  
} XrApplicationInfo;
```

### Member Descriptions

- `applicationName` is a non-empty string containing the name of the application.
- `applicationVersion` is an unsigned integer variable containing the developer-supplied version number of the application.
- `engineName` is a string containing the name of the engine (if any) used to create the application. It may be empty to indicate no specified engine.
- `engineVersion` is an unsigned integer variable containing the developer-supplied version number of the engine used to create the application. May be zero to indicate no specified engine.
- `apiVersion` is the version of this API against which the application will run, encoded as described in the [API Version Numbers and Semantics](#) section. If the runtime does not support the requested `apiVersion` it **must** return `XR_ERROR_API_VERSION_UNSUPPORTED`.

### Valid Usage (Implicit)

- `applicationName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_APPLICATION_NAME_SIZE`
- `engineName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_ENGINE_NAME_SIZE`

### Note



When using the OpenXR API to implement a reusable engine that will be used by many applications, `engineName` **should** be set to a unique string that identifies the engine, and `engineVersion` **should** encode a representation of the engine's version. This way, all applications that share this engine version will provide the same `engineName` and `engineVersion` to the runtime. The engine **should** then enable individual applications to choose their specific `applicationName` and `applicationVersion`, enabling one application to be distinguished from another application.

When using the OpenXR API to implement an individual application without a shared engine, the input `engineName` **should** be left empty and `engineVersion` **should** be set to 0. The `applicationName` **should** then be filled in with a unique string that identifies the app and the `applicationVersion` **should** encode a representation of the application's version.

The `xrDestroyInstance` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrDestroyInstance(
    XrInstance instance);
```

The `xrDestroyInstance` function is used to destroy an `XrInstance`.

### Parameter Descriptions

- `instance` is the handle to the instance to destroy.

`XrInstance` handles are destroyed using `xrDestroyInstance`. When an `XrInstance` is destroyed, all handles that are children of that `XrInstance` are also destroyed.

### Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle

### Thread Safety

- Access to `instance`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_HANDLE_INVALID`

## 4.3. Instance Information

The `xrGetInstanceProperties` function provides information about the instance and the associated runtime.

```
// Provided by XR_VERSION_1_0
XrResult xrGetInstanceProperties(
    XrInstance          instance,
    XrInstanceProperties* instanceProperties);
```

### Parameter Descriptions

- `instance` is a handle to an `XrInstance` previously created with `xrCreateInstance`.
- `instanceProperties` points to an `XrInstanceProperties` which describes the `instance`.

The `instanceProperties` parameter **must** be filled out by the runtime in response to this call, with information as defined in `XrInstanceProperties`.

### Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `instanceProperties` **must** be a pointer to an `XrInstanceProperties` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`

The `XrInstanceProperties` structure is defined as:

```
typedef struct XrInstanceProperties {  
    XrStructureType    type;  
    void*              next;  
    XrVersion          runtimeVersion;  
    char               runtimeName[XR_MAX_RUNTIME_NAME_SIZE];  
} XrInstanceProperties;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `runtimeVersion` is the runtime's version (not necessarily related to an OpenXR API version), expressed in the format of `XR_MAKE_VERSION`.
- `runtimeName` is the name of the runtime.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_INSTANCE_PROPERTIES`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 4.4. Platform-Specific Instance Creation

Some amount of data required for instance creation is exposed through chained structures defined in extensions. These structures may be **optional** or even **required** for instance creation on specific platforms, but not on other platforms. Separating off platform-specific functionality into extension structures prevents the primary `XrInstanceCreateInfo` structure from becoming too bloated with unnecessary information.

See the [List of Extensions](#) appendix for the list of available extensions and their related structures. These structures expand the `XrInstanceCreateInfo` parent struct using the `XrInstanceCreateInfo::next` member. The specific list of structures that may be used for extending `XrInstanceCreateInfo::next` can be found in the "Valid Usage (Implicit)" block immediately following the definition of the structure.

### 4.4.1. The Instance Lost Error

The `XR_ERROR_INSTANCE_LOST` error indicates that the `XrInstance` has become unusable. This **can** happen if a critical runtime process aborts, if the connection to the runtime is otherwise no longer available, or if the runtime encounters an error during any function execution which prevents it from being able to support further function execution. Once `XR_ERROR_INSTANCE_LOST` is first returned, it **must** henceforth be returned by all non-destroy functions that involve an `XrInstance` or child handle type until the instance is destroyed. Applications **must** destroy the `XrInstance`. Applications **may** then attempt to continue by recreating all relevant OpenXR objects, starting with a new `XrInstance`. A runtime **may** generate an `XrEventDataInstanceLossPending` event when instance loss is detected.

### 4.4.2. `XrEventDataInstanceLossPending`

```
// Provided by XR_VERSION_1_0
typedef struct XrEventDataInstanceLossPending {
    XrStructureType    type;
    const void*        next;
    XrTime              lossTime;
} XrEventDataInstanceLossPending;
```

Receiving the `XrEventDataInstanceLossPending` event structure indicates that the application is about to lose the indicated `XrInstance` at the indicated `lossTime` in the future. The application should call `xrDestroyInstance` and relinquish any instance-specific resources. This typically occurs to make way for a replacement of the underlying runtime, such as via a software update.

After the application has destroyed all of its instances and their children and waited past the specified time, it may then re-try `xrCreateInstance` in a loop waiting for whatever maintenance the runtime is performing to complete. The runtime will return `XR_ERROR_RUNTIME_UNAVAILABLE` from `xrCreateInstance` as long as it is unable to create the instance. Once the runtime has returned and is able to continue, it

**must** resume returning `XR_SUCCESS` from `xrCreateInstance` if valid data is passed in.

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `lossTime` is the absolute time at which the indicated instance will be considered lost and become unusable.

### Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_EVENT_DATA_INSTANCE_LOSS_PENDING`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 4.5. Instance Enumerated Type String Functions

Applications often want to turn certain enum values from the runtime into strings for use in log messages, to be localized in UI, or for various other reasons. OpenXR provides functions that turn common enum types into UTF-8 strings for use in applications.

```
// Provided by XR_VERSION_1_0
XrResult xrResultToString(
    XrInstance          instance,
    XrResult            value,
    char                buffer[XR_MAX_RESULT_STRING_SIZE]);
```

### Parameter Descriptions

- `instance` is the handle of the instance to ask for the string.
- `value` is the `XrResult` value to turn into a string.
- `buffer` is the buffer that will be used to return the string in.

Returns the text version of the provided `XrResult` value as a UTF-8 string.

In all cases the returned string **must** be one of:



## Result String Return Values

- The literal string defined for the provide numeric value in the core spec or extension. (e.g. the value 0 results in the string `XR_SUCCESS`)
- `XR_UNKNOWN_SUCCESS_` concatenated with the positive result number expressed as a decimal number.
- `XR_UNKNOWN_FAILURE_` concatenated with the negative result number expressed as a decimal number.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `value` **must** be a valid `XrResult` value
- `buffer` **must** be a character array of length `XR_MAX_RESULT_STRING_SIZE`

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`

The `xrStructureTypeToString` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrStructureTypeToString(
    XrInstance          instance,
    XrStructureType    value,
    char                buffer[XR_MAX_STRUCTURE_NAME_SIZE]);
```

## Parameter Descriptions

- `instance` is the handle of the instance to ask for the string.
- `value` is the `XrStructureType` value to turn into a string.
- `buffer` is the buffer that will be used to return the string in.

Returns the text version of the provided `XrStructureType` value as a UTF-8 string.

In all cases the returned string **must** be one of:

## Structure Type String Return Values

- The literal string defined for the provide numeric value in the core spec or extension. (e.g. the value of `XR_TYPE_INSTANCE_CREATE_INFO` results in the string `XR_TYPE_INSTANCE_CREATE_INFO`)
- `XR_UNKNOWN_STRUCTURE_TYPE_` concatenated with the structure type number expressed as a decimal number.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `value` **must** be a valid `XrStructureType` value
- `buffer` **must** be a character array of length `XR_MAX_STRUCTURE_NAME_SIZE`

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`

# Chapter 5. System

This API separates the concept of physical systems of XR devices from the logical objects that applications interact with directly. A system represents a collection of related devices in the runtime, often made up of several individual hardware components working together to enable XR experiences. An `XrSystemId` is returned by `xrGetSystem` representing the system of devices the runtime will use to support a given `form factor`. Each system may include: a VR/AR display, various forms of input (gamepad, touchpad, motion controller), and other trackable objects.

The application uses the system to create a `session`, which can then be used to accept input from the user and output rendered frames. The application also provides suggested bindings from its actions to any number of input sources. The runtime **may** use this action information to activate only a subset of devices and avoid wasting resources on devices that are not in use. Exactly which devices are active once an XR system is selected will depend on the features provided by the runtime, and **may** vary from runtime to runtime. For example, a runtime that is capable of mapping from one tracking system's space to another's **may** support devices from multiple tracking systems simultaneously.

## 5.1. Form Factors

The first step in selecting a system is for the application to request its desired **form factor**. The form factor defines how the display(s) moves in the environment relative to the user's head and how the user will interact with the XR experience. A runtime **may** support multiple form factors, such as on a mobile phone that supports both slide-in VR headset experiences and handheld AR experiences.

While an application's core XR rendering may span across form factors, its user interface will often be written to target a particular form factor, requiring explicit tailoring to function well on other form factors. For example, screen-space UI designed for a handheld phone will produce an uncomfortable experience for users if presented in screen-space on an AR headset.

```
typedef enum XrFormFactor {  
    XR_FORM_FACTOR_HEAD_MOUNTED_DISPLAY = 1,  
    XR_FORM_FACTOR_HANDHELD_DISPLAY = 2,  
    XR_FORM_FACTOR_MAX_ENUM = 0x7FFFFFFF  
} XrFormFactor;
```

The predefined form factors which **may** be supported by OpenXR runtimes are:

## Enumerant Descriptions

- `XR_FORM_FACTOR_HEAD_MOUNTED_DISPLAY`. The tracked display is attached to the user's head. The user cannot touch the display itself. A VR headset would be an example of this form factor.
- `XR_FORM_FACTOR_HANDHELD_DISPLAY`. The tracked display is held in the user's hand, independent from the user's head. The user **may** be able to touch the display, allowing for screen-space UI. A mobile phone running an AR experience using pass-through video would be an example of this form factor.

## 5.2. Getting the `XrSystemId`

```
XR_DEFINE_ATOM(XrSystemId)
```

An `XrSystemId` is an opaque atom used by the runtime to identify a system. The value `XR_NULL_SYSTEM_ID` is considered an invalid system.

```
// Provided by XR_VERSION_1_0
#define XR_NULL_SYSTEM_ID 0
```

The only `XrSystemId` value defined to be constant across all instances is the invalid system `XR_NULL_SYSTEM_ID`. No supported system is associated with `XR_NULL_SYSTEM_ID`. Unless explicitly permitted, it **should** not be passed to API calls or used as a structure attribute when a valid `XrSystemId` is required.

The `xrGetSystem` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetSystem(
    XrInstance                instance,
    const XrSystemGetInfo*    getInfo,
    XrSystemId*               systemId);
```

## Parameter Descriptions

- `instance` is the handle of the instance from which to get the information.
- `getInfo` is a pointer to an `XrSystemGetInfo` structure containing the application's requests for a system.
- `systemId` is the returned `XrSystemId`.

To get an `XrSystemId`, an application specifies its desired `form factor` to `xrGetSystem` and gets the runtime's `XrSystemId` associated with that configuration.

If the `form factor` is supported but temporarily unavailable, `xrGetSystem` **must** return `XR_ERROR_FORM_FACTOR_UNAVAILABLE`. A runtime **may** return `XR_SUCCESS` on a subsequent call for a `form factor` it previously returned `XR_ERROR_FORM_FACTOR_UNAVAILABLE`. For example, connecting or warming up hardware might cause an unavailable `form factor` to become available.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `getInfo` **must** be a pointer to a valid `XrSystemGetInfo` structure
- `systemId` **must** be a pointer to an `XrSystemId` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_FORM_FACTOR_UNSUPPORTED`
- `XR_ERROR_FORM_FACTOR_UNAVAILABLE`

The `XrSystemGetInfo` structure is defined as:

```
typedef struct XrSystemGetInfo {
    XrStructureType    type;
    const void*        next;
    XrFormFactor        formFactor;
} XrSystemGetInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `formFactor` is the [XrFormFactor](#) requested by the application.

The [XrSystemGetInfo](#) structure specifies attributes about a system as desired by an application.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SYSTEM_GET_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `formFactor` **must** be a valid [XrFormFactor](#) value

```
XrInstance instance; // previously initialized

XrSystemGetInfo system_get_info = {XR_TYPE_SYSTEM_GET_INFO};
system_get_info.formFactor = XR_FORM_FACTOR_HEAD_MOUNTED_DISPLAY;

XrSystemId systemId;
CHK_XR(xrGetSystem(instance, &system_get_info, &systemId));

// create session
// create swapchains
// begin session

// main loop

// end session
// destroy session

// no access to hardware after this point
```

## 5.3. System Properties

The `xrGetSystemProperties` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetSystemProperties(
    XrInstance          instance,
    XrSystemId         systemId,
    XrSystemProperties* properties);
```

### Parameter Descriptions

- `instance` is the instance from which `systemId` was retrieved.
- `systemId` is the `XrSystemId` whose properties will be queried.
- `properties` points to an instance of the `XrSystemProperties` structure, that will be filled with returned information.

An application **can** call `xrGetSystemProperties` to retrieve information about the system such as vendor ID, system name, and graphics and tracking properties.

### Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `properties` **must** be a pointer to an `XrSystemProperties` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SYSTEM_INVALID`

The `XrSystemProperties` structure is defined as:

```
typedef struct XrSystemProperties {
    XrStructureType    type;
    void*              next;
    XrSystemId         systemId;
    uint32_t           vendorId;
    char               systemName[XR_MAX_SYSTEM_NAME_SIZE];
    XrSystemGraphicsProperties graphicsProperties;
    XrSystemTrackingProperties trackingProperties;
} XrSystemProperties;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `vendorId` is a unique identifier for the vendor of the system.
- `systemId` is the `XrSystemId` identifying the system.
- `systemName` is a string containing the name of the system.
- `graphicsProperties` is an `XrSystemGraphicsProperties` structure specifying the system graphics properties.
- `trackingProperties` is an `XrSystemTrackingProperties` structure specifying system tracking properties.



## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_SYSTEM_PROPERTIES`
- **next** **must** be `NULL` or a valid pointer to the **next** structure in a structure chain. See also:  
`XrSystemAnchorPropertiesHTC`, `XrSystemBodyTrackingPropertiesFB`,  
`XrSystemColorSpacePropertiesFB`, `XrSystemEnvironmentDepthPropertiesMETA`,  
`XrSystemEyeGazeInteractionPropertiesEXT`, `XrSystemEyeTrackingPropertiesFB`,  
`XrSystemFaceTrackingProperties2FB`, `XrSystemFaceTrackingPropertiesFB`,  
`XrSystemFacialTrackingPropertiesHTC`, `XrSystemForceFeedbackCurlPropertiesMNDX`,  
`XrSystemFoveatedRenderingPropertiesVARJO`,  
`XrSystemFoveationEyeTrackedPropertiesMETA`,  
`XrSystemHandTrackingMeshPropertiesMSFT`, `XrSystemHandTrackingPropertiesEXT`,  
`XrSystemHeadsetIdPropertiesMETA`, `XrSystemKeyboardTrackingPropertiesFB`,  
`XrSystemMarkerTrackingPropertiesVARJO`, `XrSystemMarkerUnderstandingPropertiesML`,  
`XrSystemPassthroughColorLutPropertiesMETA`, `XrSystemPassthroughProperties2FB`,  
`XrSystemPassthroughPropertiesFB`, `XrSystemPlaneDetectionPropertiesEXT`,  
`XrSystemRenderModelPropertiesFB`, `XrSystemSpaceWarpPropertiesFB`,  
`XrSystemSpatialEntityPropertiesFB`, `XrSystemUserPresencePropertiesEXT`,  
`XrSystemVirtualKeyboardPropertiesMETA`

The runtime **must** report a valid vendor ID for the system. The vendor ID **must** be either the USB vendor ID defined for the physical device or a Khronos vendor ID.

The `XrSystemGraphicsProperties` structure is defined as:

```
typedef struct XrSystemGraphicsProperties {
    uint32_t    maxSwapchainImageHeight;
    uint32_t    maxSwapchainImageWidth;
    uint32_t    maxLayerCount;
} XrSystemGraphicsProperties;
```

## Member Descriptions

- `maxSwapchainImageHeight` is the maximum swapchain image pixel height supported by this system.
- `maxSwapchainImageWidth` is the maximum swapchain image pixel width supported by this system.
- `maxLayerCount` is the maximum number of composition layers supported by this system. The runtime **must** support at least `XR_MIN_COMPOSITION_LAYERS_SUPPORTED` layers.

```
// Provided by XR_VERSION_1_0
#define XR_MIN_COMPOSITION_LAYERS_SUPPORTED 16
```

`XR_MIN_COMPOSITION_LAYERS_SUPPORTED` defines the minimum number of composition layers that a conformant runtime must support. A runtime **must** return the `XrSystemGraphicsProperties::maxLayerCount` at least the value of `XR_MIN_COMPOSITION_LAYERS_SUPPORTED`.

The `XrSystemTrackingProperties` structure is defined as:

```
typedef struct XrSystemTrackingProperties {
    XrBool32    orientationTracking;
    XrBool32    positionTracking;
} XrSystemTrackingProperties;
```

### Member Descriptions

- `orientationTracking` is set to `XR_TRUE` to indicate the system supports orientational tracking of the view pose(s), `XR_FALSE` otherwise.
- `positionTracking` is set to `XR_TRUE` to indicate the system supports positional tracking of the view pose(s), `XR_FALSE` otherwise.

# Chapter 6. Path Tree and Semantic Paths

OpenXR incorporates an internal *semantic path tree* model, also known as the *path tree*, with entities associated with nodes organized in a logical tree and referenced by path name strings structured like a filesystem path or URL. The path tree unifies a number of concepts used in this specification and a runtime **may** add additional nodes as implementation details. As a general design principle, the most application-facing paths **should** have semantic and hierarchical meaning in their name. Thus, these paths are often referred to as *semantic paths*. However, path names in the path tree model **may** not all have the same level or kind of semantic meaning.

In regular use in an application, path name strings are converted to instance-specific `XrPath` values which are used in place of path strings. The mapping between `XrPath` values and their corresponding path name strings **may** be considered to be tracked by the runtime in a one-to-one mapping in addition to the natural tree structure of the referenced entities. Runtimes **may** use any internal implementation that satisfies the requirements.

Formally, the runtime maintains an instance-specific bijective mapping between well-formed path name strings and valid `XrPath` (`uint64_t`) values. These `XrPath` values are only valid within a single `XrInstance`, and applications **must** not share these values between instances. Applications **must** instead use the string representation of a path in their code and configuration, and obtain the correct corresponding `XrPath` at runtime in each `XrInstance`. The term *path* or *semantic path* **may** refer interchangeably to either the path name string or its associated `XrPath` value within an instance when context makes it clear which type is being discussed.

Given that path trees are a unifying model in this specification, the entities referenced by paths **can** be of diverse types. For example, they **may** be used to represent physical device or sensor *components*, which **may** be of various *component types*. They **may** also be used to represent frames of reference that are understood by the application and the runtime, as defined by an `XrSpace`. Additionally, to permit runtime re-configuration and support hardware-independent development, any syntactically-valid path string **may** be used to retrieve a corresponding `XrPath` without error given sufficient resources, *even if* no logical or hardware entity currently corresponds to that path at the time of the call. Later retrieval of the associated path string of such an `XrPath` using `xrPathToString` **should** succeed if the other requirements of that call are met. However, using such an `XrPath` in a later call to any other API function **may** result in an error if no entity of the type required by the call is available at the path at that later time. A runtime **should** permit the entity referenced by a path to vary over time to naturally reflect varying system configuration and hardware availability.

## 6.1. Path Atom Type

```
XR_DEFINE_ATOM(XrPath)
```

The `XrPath` is an atom that connects an application with a single path, within the context of a single instance. There is a bijective mapping between well-formed path strings and atoms in use. This atom is used—in place of the path name string it corresponds to—to retrieve state and perform other operations.

As an `XrPath` is only shorthand for a well-formed path string, they have no explicit life cycle.

Lifetime is implicitly managed by the `XrInstance`. An `XrPath` **must** not be used unless it is received at execution time from the runtime in the context of a particular `XrInstance`. Therefore, with the exception of `XR_NULL_PATH`, `XrPath` values **must** not be specified as constant values in applications: the corresponding path string **should** be used instead. During the lifetime of a given `XrInstance`, the `XrPath` associated with that instance with any given well-formed path **must** not vary, and similarly the well-formed path string that corresponds to a given `XrPath` in that instance **must** not vary. An `XrPath` that is received from one `XrInstance` **may** not be used with another. Such an invalid use **may** be detected and result in an error being returned, or it **may** result in undefined behavior.

Well-written applications **should** typically use a small, bounded set of paths in practice. However, the runtime **should** support looking up the `XrPath` for a large number of path strings for maximum compatibility. Runtime implementers **should** keep in mind that applications supporting diverse systems **may** look up path strings in a quantity exceeding the number of non-empty entities predicted or provided by any one runtime's own path tree model, and this is not inherently an error. However, system resources are finite and thus runtimes **may** signal exhaustion of resources dedicated to these associations under certain conditions.

When discussing the behavior of runtimes at these limits, a *new* `XrPath` refers to an `XrPath` value that, as of some point in time, has neither been received by the application nor tracked internally by the runtime. In this case, since an application has not yet received the value of such an `XrPath`, the runtime has not yet made any assertions about its association with any path string. In this context, *new* only refers to the fact that the mapping has not necessarily been made constant for a given value/path string pair for the remaining life of the associated instance by being revealed to the application. It does not necessarily imply creation of the entity, if any, referred to by such a path. Similarly, it does not imply the absence of such an entity prior to that point. Entities in the path tree have varied lifetime that is independent from the duration of the mapping from path string to `XrPath`.

For flexibility, the runtime **may** internally track or otherwise make constant, in instance or larger scope, any mapping of a path string to an `XrPath` value even before an application would otherwise receive that value, thus making it no longer *new* by the above definition.

When the runtime's resources to track the path string-`XrPath` mapping are exhausted, and the application makes an API call that would have otherwise retrieved a *new* `XrPath` as defined above, the runtime **must** return `XR_ERROR_PATH_COUNT_EXCEEDED`. This includes both explicit calls to `xrStringToPath` as well as other calls that retrieve an `XrPath` in any other way.

The runtime **should** support creating as many paths as memory will allow and **must** return `XR_ERROR_PATH_COUNT_EXCEEDED` from relevant functions when no more can be created.

```
// Provided by XR_VERSION_1_0
#define XR_NULL_PATH 0
```

The only [XPath](#) value defined to be constant across all instances is the invalid path [XR\\_NULL\\_PATH](#). No well-formed path string is associated with [XR\\_NULL\\_PATH](#). Unless explicitly permitted, it **should** not be passed to API calls or used as a structure attribute when a valid [XPath](#) is required.

## 6.2. Well-Formed Path Strings

Even though they look similar, semantic paths are not file paths. To avoid confusion with file path directory traversal conventions, many file path conventions are explicitly disallowed from well-formed path name strings.

A well-formed path name string **must** conform to the following rules:

- Path name strings **must** be constructed entirely from characters on the following list.
  - Lower case ASCII letters: a-z
  - Numeric digits: 0-9
  - Dash: -
  - Underscore: \_
  - Period: .
  - Forward Slash: /
- Path name strings **must** start with a single forward slash character.
- Path name strings **must** not end with a forward slash character.
- Path name strings **must** not contain two or more adjacent forward slash characters.
- Path name strings **must** not contain two forward slash characters that are separated by only period characters.
- Path name strings **must** not contain only period characters following the final forward slash character in the string.
- The maximum string length for a path name string, including the terminating `\0` character, is defined by [XR\\_MAX\\_PATH\\_LENGTH](#).

### 6.2.1. `xrStringToPath`

The [xrStringToPath](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrStringToPath(
    XrInstance          instance,
    const char*         pathString,
    XrPath*             path);
```

## Parameter Descriptions

- **instance** is an instance previously created.
- **pathString** is the path name string to retrieve the associated **XrPath** for.
- **path** is the output parameter, which **must** point to an **XrPath**. Given a well-formed path name string, this will be populated with an opaque value that is constant for that path string during the lifetime of that instance.

**xrStringToPath** retrieves the **XrPath** value for a well-formed path string. If such a value had not yet been assigned by the runtime to the provided path string in this **XrInstance**, one **must** be assigned at this point. All calls to this function with the same **XrInstance** and path string **must** retrieve the same **XrPath** value. Upon failure, **xrStringToPath** **must** return an appropriate **XrResult**, and **may** set the output parameter to **XR\_NULL\_PATH**. See **Path Atom Type** for the conditions under which an error **may** be returned when this function is given a valid **XrInstance** and a well-formed path string.

If the runtime's resources are exhausted and it cannot create the path, a return value of **XR\_ERROR\_PATH\_COUNT\_EXCEEDED** **must** be returned. If the application specifies a string that is not a well-formed path string, **XR\_ERROR\_PATH\_FORMAT\_INVALID** **must** be returned.



A return value of **XR\_SUCCESS** from **xrStringToPath** **may** not necessarily imply that the runtime has a component or other source of data that will be accessible through that semantic path. It only means that the path string supplied was well-formed and that the retrieved **XrPath** maps to the given path string within and during the lifetime of the **XrInstance** given.

## Valid Usage (Implicit)

- **instance** **must** be a valid **XrInstance** handle
- **pathString** **must** be a null-terminated UTF-8 string
- **path** **must** be a pointer to an **XrPath** value

## Return Codes

### Success

- XR\_SUCCESS

### Failure

- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_PATH\_FORMAT\_INVALID
- XR\_ERROR\_PATH\_COUNT\_EXCEEDED

## 6.2.2. xrPathToString

```
// Provided by XR_VERSION_1_0
XrResult xrPathToString(
    XrInstance          instance,
    XrPath              path,
    uint32_t            bufferCapacityInput,
    uint32_t*           bufferCountOutput,
    char*               buffer);
```

## Parameter Descriptions

- `instance` is an instance previously created.
- `path` is the valid `XrPath` value to retrieve the path string for.
- `bufferCapacityInput` is the capacity of the buffer, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is a pointer to the count of characters written to `buffer` (including the terminating `'\0'`), or a pointer to the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an application-allocated buffer that will be filled with the semantic path string. It **can** be `NULL` if `bufferCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

`xrPathToString` retrieves the path name string associated with an `XrPath`, in the context of a given `XrInstance`, in the form of a `NULL` terminated string placed into a *caller-allocated* buffer. Since the mapping between a well-formed path name string and an `XrPath` is bijective, there will always be exactly one string for each valid `XrPath` value. This can be useful if the calling application receives an `XrPath` value that they had not previously retrieved via `xrStringToPath`. During the lifetime of the given `XrInstance`, the path name string retrieved by this function for a given valid `XrPath` will not change. For invalid paths, including `XR_NULL_PATH`, `XR_ERROR_PATH_INVALID` **must** be returned.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values



## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_PATH_INVALID`

## 6.3. Reserved Paths

In order for some uses of semantic paths to work consistently across runtimes, it is necessary to standardize several paths and require each runtime to use the same paths or patterns of paths for certain classes of usage. Those paths are as follows.

### 6.3.1. /user paths

Some paths are used to refer to entities that are filling semantic roles in the system. These paths are all under the */user* subtree.

The reserved user paths are:

#### Reserved Semantic Paths

- */user/hand/left* represents the user's left hand. It might be tracked using a controller or other device in the user's left hand, or tracked without the user holding anything, e.g. using computer vision.
- */user/hand/right* represents the user's right hand in analog to the left hand.
- */user/head* represents inputs on the user's head, often from a device such as a head-mounted display. To reason about the user's head, see the `XR_REFERENCE_SPACE_TYPE_VIEW` reference space.
- */user/gamepad* is a two-handed gamepad device held by the user.
- */user/treadmill* is a treadmill or other locomotion-targeted input device.

Runtimes are not required to provide interaction at all of these paths. For instance, in a system with no

hand tracking, only `/user/head` would be active for interaction. In a system with only one controller, the runtime **may** provide access to that controller via either `/user/hand/left` or `/user/hand/right` as it deems appropriate.

The runtime **may** change the devices referred to by `/user/hand/left` and `/user/hand/right` at any time.

If more than two hand-held controllers or devices are active, the runtime **must** determine which two are accessible as `/user/hand/left` and `/user/hand/right`.

### 6.3.2. Input subpaths

Devices on the source side of the input system need to define paths for each component that can be bound to an action. This section describes the naming conventions for those input components. Runtimes **must** ignore input source paths that use identifiers and component names that do not appear in this specification or otherwise do not follow the pattern specified below.

Each input source path **must** match the following pattern:

- `.../input/<identifier>[_<location>][/<component>]`

Identifiers are often the label on the component or related to the type and location of the component.

When specifying a suggested binding there are several cases where the component part of the path can be determined automatically. See [Suggested Bindings](#) for more details.

See [Interaction Profiles](#) for examples of input subpaths.

#### Standard identifiers

- trackpad - A 2D input source that usually includes click and touch component.
- thumbstick - A small 2D joystick that is meant to be used with the user's thumb. These sometimes include click and/or touch components.
- joystick - A 2D joystick that is meant to be used with the user's entire hand, such as a flight stick. These generally do not have click component, but might have touch components.
- trigger - A 1D analog input component that returns to a rest state when the user stops interacting with it. These sometime include touch and/or click components.
- throttle - A 1D analog input component that remains in position when the user stops interacting with it.
- trackball - A 2D relative input source. These sometimes include click components.
- pedal - A 1D analog input component that is similar to a trigger but meant to be operated by a foot
- system - A button with the specialised meaning that it enables the user to access system-level functions and UI. Input data from system buttons is generally used internally by runtimes and **may** not be available to applications.
- dpad\_up, dpad\_down, dpad\_left, and dpad\_right - A set of buttons arranged in a plus shape.

- `diamond_up`, `diamond_down`, `diamond_left`, and `diamond_right` - Gamepads often have a set of four buttons arranged in a diamond shape. The labels on those buttons vary from gamepad to gamepad, but their arrangement is consistent. These names are used for the A/B/X/Y buttons on a Xbox controller, and the square/cross/circle/triangle button on a PlayStation controller.
- `a`, `b`, `x`, `y`, `start`, `home`, `end`, `select` - Standalone buttons are named for their physical labels. These are the standard identifiers for such buttons. Extensions **may** add new identifiers as detailed in the next section. Groups of four buttons in a diamond shape **should** use the diamond-prefix names above instead of using the labels on the buttons themselves.
- `volume_up`, `volume_down`, `mute_mic`, `play_pause`, `menu`, `view`, `back` - Some other standard controls are often identified by icons. These are their standard names.
- `thumbrest` - Some controllers have a place for the user to rest their thumb.
- `shoulder` - A button that is usually pressed with the index finger and is often positioned above a trigger.
- `squeeze` - An input source that indicates that the user is squeezing their fist closed. This could be a simple button or act more like a trigger. Sources with this identifier **should** either follow button or trigger conventions for their components.
- `wheel` - A steering wheel.
- `thumb_resting_surfaces` - Any surfaces that a thumb may naturally rest on. This may include, but is not limited to, face buttons, thumbstick, and thumbrest (Provided by `XR_VERSION_1_1`)
- `stylus` - Tip that can be used for writing or drawing. May be able to detect various pressure levels (Provided by `XR_VERSION_1_1`)
- `trigger_curl` - This sensor detects how pointed or curled the user's finger is on the trigger: 0 = fully pointed, 1 = finger flat on surface (Provided by `XR_VERSION_1_1`)
- `trigger_slide` - This sensor represents how far the user is sliding their index finger along the surface of the trigger: 0 = finger flat on the surface, 1 = finger fully drawn back (Provided by `XR_VERSION_1_1`)

## Standard pose identifiers

Input sources whose orientation and/or position are tracked also expose pose identifiers.

Standard pose identifiers for tracked hands or motion controllers as represented by `/user/hand/left` and `/user/hand/right` are:

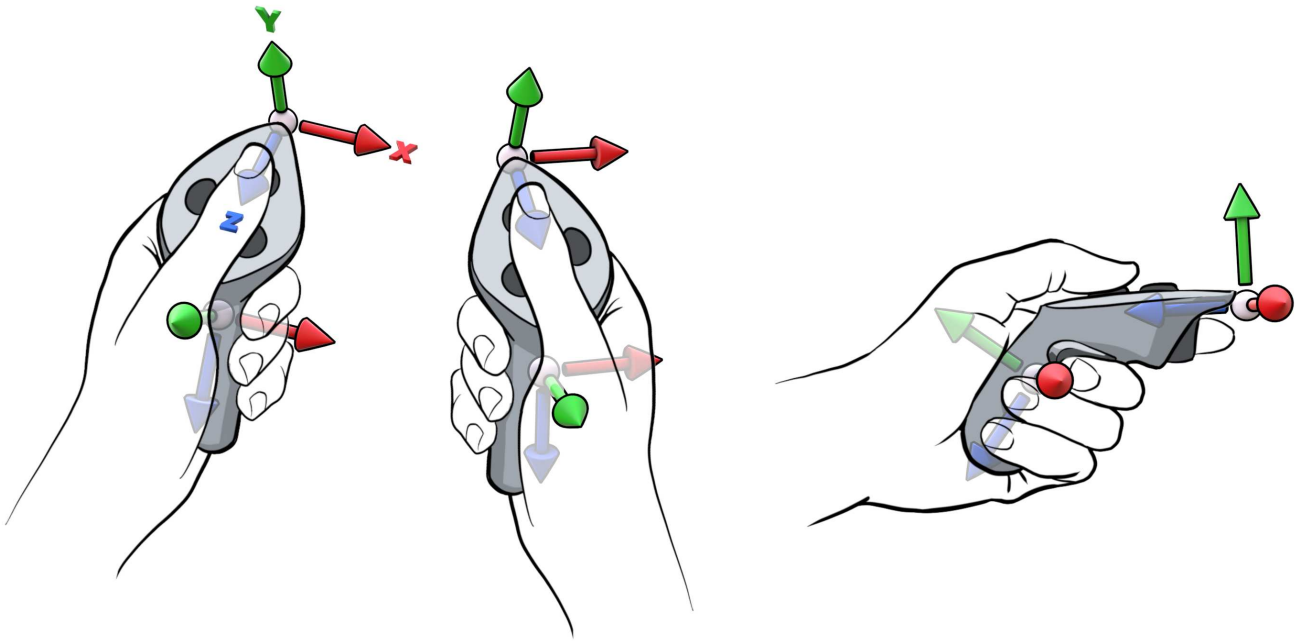


Figure 2. Example grip and aim poses for generic motion controllers

- grip - A pose that allows applications to reliably render a virtual object held in the user's hand, whether it is tracked directly or by a motion controller. The grip pose is defined as follows:
  - The grip position:
    - For tracked hands: The user's palm centroid when closing the fist, at the surface of the palm.
    - For handheld motion controllers: A fixed position within the controller that generally lines up with the palm centroid when held by a hand in a neutral position. This position should be adjusted left or right to center the position within the controller's grip.
  - The grip orientation's +X axis: When you completely open your hand to form a flat 5-finger pose, the ray that is normal to the user's palm (away from the palm in the left hand, into the palm in the right hand).
  - The grip orientation's -Z axis: When you close your hand partially (as if holding the controller), the ray that goes through the center of the tube formed by your non-thumb fingers, in the direction of little finger to thumb.
  - The grip orientation's +Y axis: orthogonal to +Z and +X using the right-hand rule.
- aim - A pose that allows applications to point in the world using the input source, according to the platform's conventions for aiming with that kind of source. The aim pose is defined as follows:
  - For tracked hands: The ray that follows platform conventions for how the user aims at objects in the world with their entire hand, with +Y up, +X to the right, and -Z forward. The ray chosen will be runtime-dependent, often a ray emerging from the hand at a target pointed by moving the forearm.

- For handheld motion controllers: The ray that follows platform conventions for how the user targets objects in the world with the motion controller, with +Y up, +X to the right, and -Z forward. This is usually for applications that are rendering a model matching the physical controller, as an application rendering a virtual object in the user's hand likely prefers to point based on the geometry of that virtual object. The ray chosen will be runtime-dependent, although this will often emerge from the frontmost tip of a motion controller.
- grip\_surface - (Provided by XR\_VERSION\_1\_1) A pose that allows applications to reliably anchor visual content relative to the user's physical hand, whether the user's hand is tracked directly or its position and orientation is inferred by a physical controller. The grip\_surface pose is defined as follows:
  - The grip\_surface position: The user's physical palm centroid, at the surface of the palm. For the avoidance of doubt, the palm does not include fingers.
  - The grip\_surface orientation's +X axis: When a user is holding the controller and straightens their index fingers pointing forward, the ray that is normal (perpendicular) to the user's palm (away from the palm in the left hand, into the palm in the right hand).
  - The grip\_surface orientation's -Z axis: When a user is holding the controller and straightens their index finger, the ray that is parallel to their finger's pointing direction.
  - The grip\_surface orientation's +Y axis: orthogonal to +Z and +X using the right-hand rule.

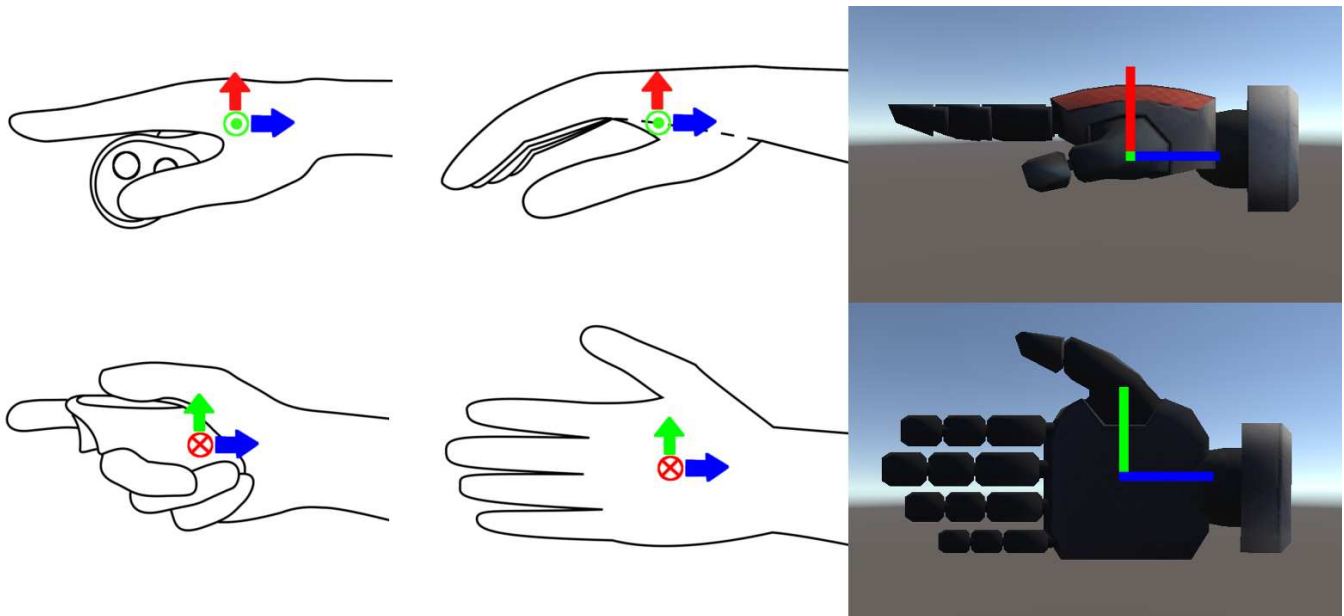


Figure 3. Example grip\_surface pose for (from left to right) a generic motion controller, tracked hand, and a digital hand avatar. The X axis is depicted in red. The Y axis is depicted in green. The Z axis is depicted in blue.

Note



When the `XR_EXT_palm_pose` extension is available and enabled, an additional "palm\_ext" standard pose identifier is available, and a path is added to all interaction profiles valid for `/user/hand/left` or `/user/hand/right`. This includes interaction profiles defined in the core spec and in extensions.



### Note

When the `XR_EXT_hand_interaction` extension is available and enabled, additional "pinch\_ext" and "poke\_ext" standard pose identifiers are available, and a path is added to all interaction profiles valid for `/user/hand/left` or `/user/hand/right`. This includes interaction profiles defined in the core spec and in extensions.

## Standard locations

When a single device contains multiple input sources that use the same identifier, a location suffix is added to create a unique identifier for that input source.

Standard locations are:

- left
- right
- left\_upper
- left\_lower
- right\_upper
- right\_lower
- upper
- lower

## Standard components

Components are named for the specific boolean, scalar, or other value of the input source. Standard components are:

- click - A physical switch has been pressed by the user. This is valid for all buttons, and is common for trackpads, thumbsticks, triggers, and dpads. "click" components are always boolean.
- touch - The user has touched the input source. This is valid for all trackpads, and **may** be present for any other kind of input source if the device includes the necessary sensor. "touch" components are always boolean.
- force - A 1D scalar value that represents the user applying force to the input. It varies from 0 to 1, with 0 being the rest state. This is present for any input source with a force sensor.
- value - A 1D scalar value that varies from 0 to 1, with 0 being the rest state. This is present for triggers, throttles, and pedals. It **may** also be present for squeeze or other components.
- x, y - scalar components of 2D values. These vary in value from -1 to 1. These represent the 2D position of the input source with 0 being the rest state on each axis. -1 means all the way left for x axis or all the way down for y axis. +1 means all the way right for x axis or all the way up for y axis. x and y components are present for trackpads, thumbsticks, and joysticks.
- twist - Some sources, such as flight sticks, have a sensor that allows the user to twist the input left

or right. For this component -1 means all the way left and 1 means all the way right.

- pose - The orientation and/or position of this input source. This component **may** exist for dedicated pose identifiers like grip and aim, or **may** be defined on other identifiers such as trackpad to let applications reason about the surface of that part.
- proximity - The user is in physical proximity of input source. This **may** be present for any kind of input source representing a physical component, such as a button, if the device includes the necessary sensor. The state of a "proximity" component **must** be `XR_TRUE` if the same input source is returning `XR_TRUE` for either a "touch" or any other component that implies physical contact. The runtime **may** return `XR_TRUE` for "proximity" when "touch" returns `XR_FALSE` which would indicate that the user is hovering just above, but not touching the input source in question. "proximity" components are always boolean. (Provided by `XR_VERSION_1_1`)

## Output paths

Many devices also have subpaths for output features such as haptics. The runtime **must** ignore output component paths that do not follow the pattern:

- `.../output/<output_identifier>[_<location>]`

Standard output identifiers are:

- haptic - A haptic element like an LRA (Linear Resonant Actuator) or vibration motor
- haptic\_trigger - A haptic element located in the trigger (Provided by `XR_VERSION_1_1`)
- haptic\_thumb - A haptic element located in the resting place of the thumb, like under the touchpad (Provided by `XR_VERSION_1_1`)

Devices which contain multiple haptic elements with the same output identifier must use a location suffix as specified above.

### 6.3.3. Adding input sources via extensions

Extensions **may** enable input source path identifiers, output source path identifiers, and component names that are not included in the core specification, subject to the following conditions:

- EXT extensions **must** include the `_ext` suffix on any identifier or component name. E.g. `.../input/newidentifier_ext/newcomponent_ext`
- Vendor extensions **must** include the vendor's tag as a suffix on any identifier or component name. E.g. `.../input/newidentifier_vendor/newcomponent_vendor` (where "vendor" is replaced with the vendor's actual extension tag.)
- Khronos (KHR) extensions **may** add undecorated identifier or component names.

These rules are in place to prevent extensions from adding first class undecorated names that become defacto standards. Runtimes **must** ignore input source paths that do not follow the restrictions above.

Extensions **may** also add new location suffixes, and **may** do so by adding a new identifier and location combination using the appropriate suffix. E.g. `.../input/newidentifier_newlocation_ext`

## 6.4. Interaction Profile Paths

An interaction profile path identifies a collection of buttons and other input sources in a physical arrangement to allow applications and runtimes to coordinate action bindings.

Interaction profile paths are of the form:

- `/interaction_profiles/<vendor_name>/<type_name>`



### Note

When the `XR_EXT_palm_pose` extension is available and enabled, an additional input component path is added to all core interaction profiles valid for `/user/hand/left` or `/user/hand/right`. See the extension for more details.

### 6.4.1. Khronos Simple Controller Profile

Path: `/interaction_profiles/khr/simple_controller`

Valid for user paths:

- `/user/hand/left`
- `/user/hand/right`

This interaction profile provides basic pose, button, and haptic support for applications with simple input needs. There is no hardware associated with the profile, and runtimes which support this profile **should** map the input paths provided to whatever the appropriate paths are on the actual hardware.

Supported component paths:

- `.../input/select/click`
- `.../input/menu/click`
- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../output/haptic`



### Note

When the runtime supports `XR_VERSION_1_1` and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`





*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## 6.4.2. Bytedance PICO Neo 3 controller Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/bytedance/pico\_neo3\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Bytedance PICO Neo3 Controller.

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*

- *.../input/b/click*
- *.../input/b/touch*
- *.../input/menu/click*
- *.../input/system/click* (**may** not be available for application use)
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/thumbstick/y*
- *.../input/thumbstick/x*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/squeeze/click*
- *.../input/squeeze/value*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



#### Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

### 6.4.3. Bytedance PICO 4 controller Profile

(Provided by `XR_VERSION_1_1`)

Path: `/interaction_profiles/bytedance/pico4_controller`

Valid for user paths:

- `/user/hand/left`
- `/user/hand/right`

This interaction profile represents the input sources and haptics on the Bytedance PICO 4 Controller.

- On `/user/hand/left` only:
  - `.../input/x/click`
  - `.../input/x/touch`
  - `.../input/y/click`
  - `.../input/y/touch`
  - `.../input/menu/click`
- On `/user/hand/right` only:
  - `.../input/a/click`
  - `.../input/a/touch`
  - `.../input/b/click`
  - `.../input/b/touch`
- `.../input/system/click` (**may** not be available for application use)
- `.../input/trigger/click`
- `.../input/trigger/value`
- `.../input/trigger/touch`
- `.../input/thumbstick/y`
- `.../input/thumbstick/x`
- `.../input/thumbstick/click`

- *.../input/thumbstick/touch*
- *.../input/squeeze/click*
- *.../input/squeeze/value*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

#### 6.4.4. Bytedance PICO G3 controller Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/bytedance/pico\_g3\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Bytedance PICO G3 Controller.

- `.../input/trigger/click`
- `.../input/trigger/value`
- `.../input/menu/click`
- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../input/thumbstick`
- `.../input/thumbstick/click`

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

*Note*



When designing suggested bindings for this interaction profile, you **may** suggest bindings for both `/user/hand/left` and `/user/hand/right`. However, only one of them will be active at a given time, so do not design interactions that require simultaneous use of both hands.

## 6.4.5. Google Daydream Controller Profile

Path: `/interaction_profiles/google/daydream_controller`

Valid for user paths:

- `/user/hand/left`
- `/user/hand/right`

This interaction profile represents the input sources on the Google Daydream Controller.

Supported component paths:

- `.../input/select/click`
- `.../input/trackpad/x`
- `.../input/trackpad/y`
- `.../input/trackpad/click`
- `.../input/trackpad/touch`
- `.../input/grip/pose`
- `.../input/aim/pose`

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



#### Note

When the `XR_EXT_hand_interaction` extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

### 6.4.6. HP Mixed Reality Motion Controller Profile

(Provided by `XR_VERSION_1_1`)

Path: `/interaction_profiles/hp/mixed_reality_controller`

Valid for user paths:

- `/user/hand/left`
- `/user/hand/right`

This interaction profile represents the input sources and haptics on the HP Mixed Reality Motion Controller.

- On `/user/hand/left` only:
  - `.../input/x/click`
  - `.../input/y/click`
- On `/user/hand/right` only:
  - `.../input/a/click`
  - `.../input/b/click`
- `.../input/menu/click`
- `.../input/squeeze/value`
- `.../input/trigger/value`
- `.../input/thumbstick/x`
- `.../input/thumbstick/y`
- `.../input/thumbstick/click`
- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../output/haptic`



Note

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



Note

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



Note

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

### 6.4.7. HTC Vive Controller Profile

Path: */interaction\_profiles/htc/vive\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Vive Controller.

Supported component paths:

- *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/click*
- *.../input/menu/click*
- *.../input/trigger/click*



- *.../input/trigger/value*
- *.../input/trackpad/x*
- *.../input/trackpad/y*
- *.../input/trackpad/click*
- *.../input/trackpad/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## 6.4.8. HTC Vive Cosmos Controller Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/htc/vive\_cosmos\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Vive Cosmos Controller.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/y/click*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/b/click*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/shoulder/click*
- *.../input/squeeze/click*
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



Note

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



Note

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

### 6.4.9. HTC Vive Focus 3 Controller Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/htc/vive\_focus3\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Vive Focus 3 Controller.

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/y/click*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/b/click*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/click*

- *.../input/squeeze/touch*
- *.../input/squeeze/value*
- *.../input/trigger/click*
- *.../input/trigger/touch*
- *.../input/trigger/value*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/thumbrest/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

### 6.4.10. HTC Vive Pro Profile

Path: */interaction\_profiles/htc/vive\_pro*

Valid for user paths:

- */user/head*

This interaction profile represents the input sources on the Vive Pro headset.

Supported component paths:

- *.../input/system/click* (**may** not be available for application use)
- *.../input/volume\_up/click*
- *.../input/volume\_down/click*
- *.../input/mute\_mic/click*

### 6.4.11. Magic Leap 2 Controller Profile

(Provided by `XR_VERSION_1_1`)

Path: */interaction\_profiles/ml/ml2\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Magic Leap 2 controller.

Supported component paths:

- *.../input/menu/click*
- *.../input/home/click* (**may** not be available for application use)
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/trackpad/y*
- *.../input/trackpad/x*
- *.../input/trackpad/click*
- *.../input/trackpad/force*
- *.../input/trackpad/touch*
- *.../input/aim/pose*

- *.../input/grip/pose*
- *.../input/shoulder/click*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## 6.4.12. Microsoft Mixed Reality Motion Controller Profile

Path: */interaction\_profiles/microsoft/motion\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Microsoft Mixed Reality Controller.

Supported component paths:

- *.../input/menu/click*
- *.../input/squeeze/click*
- *.../input/trigger/value*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/trackpad/x*
- *.../input/trackpad/y*
- *.../input/trackpad/click*
- *.../input/trackpad/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

### 6.4.13. Microsoft Xbox Controller Profile

Path: */interaction\_profiles/microsoft/xbox\_controller*

Valid for user paths:

- */user/gamepad*

This interaction profile represents the input sources and haptics on the Microsoft Xbox Controller.

Supported component paths:

- *.../input/menu/click*
- *.../input/view/click*
- *.../input/a/click*
- *.../input/b/click*
- *.../input/x/click*
- *.../input/y/click*
- *.../input/dpad\_down/click*
- *.../input/dpad\_right/click*
- *.../input/dpad\_up/click*
- *.../input/dpad\_left/click*
- *.../input/shoulder\_left/click*
- *.../input/shoulder\_right/click*
- *.../input/thumbstick\_left/click*
- *.../input/thumbstick\_right/click*
- *.../input/trigger\_left/value*
- *.../input/trigger\_right/value*
- *.../input/thumbstick\_left/x*
- *.../input/thumbstick\_left/y*
- *.../input/thumbstick\_right/x*
- *.../input/thumbstick\_right/y*
- *.../output/haptic\_left*
- *.../output/haptic\_right*
- *.../output/haptic\_left\_trigger*
- *.../output/haptic\_right\_trigger*



## 6.4.14. Oculus Go Controller Profile

Path: */interaction\_profiles/oculus/go\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources on the Oculus Go controller.

Supported component paths:

- *.../input/system/click* (**may** not be available for application use)
- *.../input/trigger/click*
- *.../input/back/click*
- *.../input/trackpad/x*
- *.../input/trackpad/y*
- *.../input/trackpad/click*
- *.../input/trackpad/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*

### Note



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

### Note



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

### Note



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



#### Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

### 6.4.15. Oculus Touch Controller Profile

Path: `/interaction_profiles/oculus/touch_controller`

Valid for user paths:

- `/user/hand/left`
- `/user/hand/right`

This interaction profile represents the input sources and haptics on the Oculus Touch controller.

Supported component paths:

- On `/user/hand/left` only:
  - `.../input/x/click`
  - `.../input/x/touch`
  - `.../input/y/click`
  - `.../input/y/touch`
  - `.../input/menu/click`
- On `/user/hand/right` only:
  - `.../input/a/click`
  - `.../input/a/touch`
  - `.../input/b/click`
  - `.../input/b/touch`
  - `.../input/system/click` (**may** not be available for application use)
- `.../input/squeeze/value`
- `.../input/trigger/value`
- `.../input/trigger/touch`
- `.../input/trigger/proximity` (Provided by `XR_VERSION_1_1`)
- `.../input/thumb_resting_surfaces/proximity` (Provided by `XR_VERSION_1_1`)
- `.../input/thumbstick/x`

- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/thumbrest/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## 6.4.16. Meta Touch Pro Controller Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/meta/touch\_pro\_controller*

Valid for user paths:

- */user/hand/left*

- */user/hand/right*

This interaction profile represents the input sources and haptics on the Meta Touch Pro controller.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/value*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/trigger/proximity*
- *.../input/trigger\_curl/value*
- *.../input/trigger\_slide/value*
- *.../input/thumb\_resting\_surfaces/proximity*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/thumbrest/touch*
- *.../input/thumbrest/force*
- *.../input/stylus/force*
- *.../input/grip/pose*
- *.../input/aim/pose*

- `.../output/haptic`
- `.../output/haptic_trigger`
- `.../output/haptic_thumb`

*Note*



When the runtime supports `XR_VERSION_1_1` and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the `XR_KHR_maintenance1` extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the `XR_EXT_palm_pose` extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`

*Note*



When the `XR_EXT_hand_interaction` extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

### 6.4.17. Meta Touch Plus Controller Profile

(Provided by `XR_VERSION_1_1`)

Path: `/interaction_profiles/meta/touch_plus_controller`

Valid for user paths:

- `/user/hand/left`
- `/user/hand/right`

This interaction profile represents the input sources and haptics on the Meta Touch Plus controller.

Supported component paths:

- On `/user/hand/left` only:
  - `.../input/x/click`
  - `.../input/x/touch`
  - `.../input/y/click`
  - `.../input/y/touch`
  - `.../input/menu/click`
- On `/user/hand/right` only:
  - `.../input/a/click`
  - `.../input/a/touch`
  - `.../input/b/click`
  - `.../input/b/touch`
  - `.../input/system/click` (**may** not be available for application use)
- `.../input/squeeze/value`
- `.../input/trigger/value`
- `.../input/trigger/touch`
- `.../input/trigger/force`
- `.../input/trigger/proximity`
- `.../input/trigger_curl/value`
- `.../input/trigger_slide/value`
- `.../input/thumb_resting_surfaces/proximity`
- `.../input/thumbstick/x`
- `.../input/thumbstick/y`
- `.../input/thumbstick/click`
- `.../input/thumbstick/touch`
- `.../input/thumbrest/touch`
- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../output/haptic`

*Note*



When the runtime supports `XR_VERSION_1_1` and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`



*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## 6.4.18. Meta Touch Controller (Rift CV1) Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/meta/touch\_controller\_rift\_cv1*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Oculus Touch controller and is a legacy profile added to specifically represent the controller shipped with the Rift CV1.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:

- *.../input/a/click*
- *.../input/a/touch*
- *.../input/b/click*
- *.../input/b/touch*
- *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/value*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/trigger/proximity*
- *.../input/thumb\_resting\_surfaces/proximity*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/thumbrest/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*





#### Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

### 6.4.19. Meta Touch Controller (Rift S / Quest 1) Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/meta/touch\_controller\_quest\_1\_rift\_s*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Oculus Touch controller and is a legacy profile added to specifically represent the controller shipped with the Rift S and Quest 1.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/value*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/trigger/proximity*

- `.../input/thumb_resting_surfaces/proximity`
- `.../input/thumbstick/x`
- `.../input/thumbstick/y`
- `.../input/thumbstick/click`
- `.../input/thumbstick/touch`
- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../output/haptic`

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

## 6.4.20. Meta Touch Controller (Quest 2) Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: `/interaction_profiles/meta/touch_controller_quest_2`

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Oculus Touch controller and is a legacy profile added to specifically represent the controller shipped with the Quest 2.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/value*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/trigger/proximity*
- *.../input/thumb\_resting\_surfaces/proximity*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/thumbrest/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*



*Note*

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## 6.4.21. Samsung Odyssey Controller Profile

(Provided by [XR\\_VERSION\\_1\\_1](#))

Path: */interaction\_profiles/samsung/odyssey\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Samsung Odyssey Controller. It is exactly the same, with the exception of the name of the interaction profile, as the Microsoft Mixed Reality Controller interaction profile. It enables the application to differentiate the newer form factor of motion controller released with the Samsung Odyssey headset. It enables the application to customize the appearance and experience of the controller differently from the original [mixed reality motion controller](#).

Supported component paths:

- `.../input/menu/click`
- `.../input/squeeze/click`
- `.../input/trigger/value`
- `.../input/thumbstick/x`
- `.../input/thumbstick/y`
- `.../input/thumbstick/click`
- `.../input/trackpad/x`
- `.../input/trackpad/y`
- `.../input/trackpad/click`
- `.../input/trackpad/touch`
- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../output/haptic`

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

## 6.4.22. Valve Index Controller Profile

Path: */interaction\_profiles/valve/index\_controller*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Valve Index controller.

Supported component paths:

- *.../input/system/click* (**may** not be available for application use)
- *.../input/system/touch* (**may** not be available for application use)
- *.../input/a/click*
- *.../input/a/touch*
- *.../input/b/click*
- *.../input/b/touch*
- *.../input/squeeze/value*
- *.../input/squeeze/force*
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/trackpad/x*
- *.../input/trackpad/y*
- *.../input/trackpad/force*
- *.../input/trackpad/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*



*Note*

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

# Chapter 7. Spaces

Across both virtual reality and augmented reality, XR applications have a core need to map the location of virtual objects to the corresponding real-world locations where they will be rendered. **Spaces** allow applications to explicitly create and specify the frames of reference in which they choose to track the real world, and then determine how those frames of reference move relative to one another over time.

```
XR_DEFINE_HANDLE(XrSpace)
```

Spaces are represented by [XrSpace](#) handles, which the application creates and then uses in API calls. Whenever an application calls a function that returns coordinates, it provides an [XrSpace](#) to specify the frame of reference in which those coordinates will be expressed. Similarly, when providing coordinates to a function, the application specifies which [XrSpace](#) the runtime should use to interpret those coordinates.

OpenXR defines a set of well-known **reference spaces** that applications use to bootstrap their spatial reasoning. These reference spaces are: **VIEW**, **LOCAL**, **LOCAL\_FLOOR**, and **STAGE**. Each reference space has a well-defined meaning, which establishes where its origin is positioned and how its axes are oriented.

Runtimes whose tracking systems improve their understanding of the world over time **may** track spaces independently. For example, even though a **LOCAL space** and a **STAGE space** each map their origin to a static position in the world, a runtime with an inside-out tracking system **may** introduce slight adjustments to the origin of each space on a continuous basis to keep each origin in place.

Beyond well-known reference spaces, runtimes expose other independently-tracked spaces, such as a pose action space that tracks the pose of a motion controller over time.

When one or both spaces are tracking a dynamic object, passing in an updated time to [xrLocateSpace](#) each frame will result in an updated relative pose. For example, the location of the left hand's pose action space in the **STAGE** reference space will change each frame as the user's hand moves relative to the stage's predefined origin on the floor. In other XR APIs, it is common to report the "pose" of an object relative to some presumed underlying global space. This API is careful to not explicitly define such an underlying global space, because it does not apply to all systems. Some systems will support no **STAGE space**, while others may support a **STAGE space** that switches between various physical stages with dynamic availability. To satisfy this wide variability, "poses" are always described as the relationship between two spaces.

Some devices improve their understanding of the world as the device is used. The location returned by [xrLocateSpace](#) in later frames **may** change over time, even for spaces that track static objects, as either the target space or base space adjusts its origin.

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.

## 7.1. Reference Spaces

The `XrReferenceSpaceType` enumeration is defined as:

```
typedef enum XrReferenceSpaceType {
    XR_REFERENCE_SPACE_TYPE_VIEW = 1,
    XR_REFERENCE_SPACE_TYPE_LOCAL = 2,
    XR_REFERENCE_SPACE_TYPE_STAGE = 3,
    // Provided by XR_VERSION_1_1
    XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR = 1000426000,
    // Provided by XR_MSFT_unbounded_reference_space
    XR_REFERENCE_SPACE_TYPE_UNBOUNDED_MSFT = 1000038000,
    // Provided by XR_VARJO_foveated_rendering
    XR_REFERENCE_SPACE_TYPE_COMBINED_EYE_VARJO = 1000121000,
    // Provided by XR_ML_localization_map
    XR_REFERENCE_SPACE_TYPE_LOCALIZATION_MAP_ML = 1000139000,
    // Provided by XR_EXT_local_floor
    XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR_EXT = XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR,
    XR_REFERENCE_SPACE_TYPE_MAX_ENUM = 0x7FFFFFFF
} XrReferenceSpaceType;
```

Brief introductions to core reference space types follow. Each has full requirements in a subsequent section, linked from these descriptions.

## Enumerant Descriptions

- `XR_REFERENCE_SPACE_TYPE_VIEW`. The `VIEW` reference space tracks the view origin used to generate view transforms for the primary viewer (or centroid of view origins if stereo), with +Y up, +X to the right, and -Z forward. This space points in the forward direction for the viewer without incorporating the user's eye orientation, and is not gravity-aligned.

Runtimes **must** support `VIEW` reference space.

- `XR_REFERENCE_SPACE_TYPE_LOCAL`. The `LOCAL` reference space establishes a world-locked origin, gravity-aligned to exclude pitch and roll, with +Y up, +X to the right, and -Z forward. This space locks in both its initial position and orientation, which the runtime **may** define to be either the initial position at application launch or some other calibrated zero position.

Runtimes **must** support `LOCAL` reference space.

- `XR_REFERENCE_SPACE_TYPE_STAGE`. The `STAGE` reference space is a runtime-defined flat, rectangular space that is empty and can be walked around on. The origin is on the floor at the center of the rectangle, with +Y up, and the X and Z axes aligned with the rectangle edges. The runtime **may** not be able to locate spaces relative to the `STAGE` reference space if the user has not yet defined one within the runtime-specific UI. Applications **can** use `xrGetReferenceSpaceBoundsRect` to determine the extents of the `STAGE` reference space's XZ bounds rectangle, if defined.

Support for the `STAGE` reference space is **optional**.

- `XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR` (provided by `XR_VERSION_1_1`) Similar to `LOCAL` space, the `LOCAL_FLOOR` reference space establishes a world-locked origin, gravity-aligned to exclude pitch and roll, with +Y up, +X to the right, and -Z forward. However, the origin of this space is defined to be on an estimate of the floor level.

Runtimes **must** support `LOCAL_FLOOR` reference space.

An `XrSpace` handle for a reference space is created using `xrCreateReferenceSpace`, by specifying the chosen reference space type and a pose within the natural reference frame defined for that reference space type.

Runtimes implement well-known reference spaces from `XrReferenceSpaceType` if they support tracking of that kind. Available reference space types are indicated by `xrEnumerateReferenceSpaces`. Note that other spaces can be created as well, such as pose action spaces created by `xrCreateActionSpace`, which are not enumerated by that API.

### 7.1.1. View Reference Space

The `XR_REFERENCE_SPACE_TYPE_VIEW` or `VIEW` reference space tracks the view origin used to generate view

transforms for the primary viewer (or centroid of view origins if stereo), with +Y up, +X to the right, and -Z forward. This space points in the forward direction for the viewer without incorporating the user's eye orientation, and is not gravity-aligned.

The **VIEW space** is primarily useful when projecting from the user's perspective into another space to obtain a targeting ray, or when rendering small head-locked content such as a reticle. Content rendered in the **VIEW space** will stay at a fixed point on head-mounted displays and may be uncomfortable to view if too large. To obtain the ideal view and projection transforms to use each frame for rendering world content, applications should call `xrLocateViews` instead of using this space.

### 7.1.2. Local Reference Space

The `XR_REFERENCE_SPACE_TYPE_LOCAL` or **LOCAL** reference space establishes a world-locked origin, gravity-aligned to exclude pitch and roll, with +Y up, +X to the right, and -Z forward. This space locks in both its initial position and orientation, which the runtime **may** define to be either the initial position at application launch or some other calibrated zero position.

When a user needs to recenter the **LOCAL space**, a runtime **may** offer some system-level recentering interaction that is transparent to the application, but which causes the current leveled head space to become the new **LOCAL** space. When such a recentering occurs, the runtime **must** queue the `XrEventDataReferenceSpaceChangePending` event, with the recentered **LOCAL** space origin only taking effect for `xrLocateSpace` or `xrLocateViews` calls whose `XrTime` parameter is greater than or equal to the `XrEventDataReferenceSpaceChangePending::changeTime` in that event.

When views, controllers or other spaces experience tracking loss relative to the **LOCAL space**, runtimes **should** continue to provide inferred or last-known `position` and `orientation` values. These inferred poses can, for example, be based on neck model updates, inertial dead reckoning, or a last-known position, so long as it is still reasonable for the application to use that pose. While a runtime is providing position data, it **must** continue to set `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_VIEW_STATE_POSITION_VALID_BIT` but it **can** clear `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` and `XR_VIEW_STATE_POSITION_TRACKED_BIT` to indicate that the position is inferred or last-known in this way.

When tracking is recovered, runtimes **should** snap the pose of other spaces back into position relative to the original origin of **LOCAL space**.

### 7.1.3. Stage Reference Space

The **STAGE** reference space is a runtime-defined flat, rectangular space that is empty and can be walked around on. The origin is on the floor at the center of the rectangle, with +Y up, and the X and Z axes aligned with the rectangle edges. The runtime **may** not be able to locate spaces relative to the **STAGE** reference space if the user has not yet defined one within the runtime-specific UI. Applications **can** use `xrGetReferenceSpaceBoundsRect` to determine the extents of the **STAGE** reference space's XZ bounds rectangle, if defined.

The **STAGE space** is useful when an application needs to render **standing-scale** content (no bounds) or **room-scale** content (with bounds) that is relative to the physical floor.

When the user redefines the origin or bounds of the current **STAGE space**, or the runtime otherwise switches to a new **STAGE** space definition, the runtime **must** queue the `XrEventDataReferenceSpaceChangePending` event, with the new **STAGE** space origin only taking effect for `xrLocateSpace` or `xrLocateViews` calls whose `XrTime` parameter is greater than or equal to the `XrEventDataReferenceSpaceChangePending::changeTime` in that event.

When views, controllers, or other spaces experience tracking loss relative to the **STAGE space**, runtimes **should** continue to provide inferred or last-known **position** and **orientation** values. These inferred poses can, for example, be based on neck model updates, inertial dead reckoning, or a last-known position, so long as it is still reasonable for the application to use that pose. While a runtime is providing position data, it **must** continue to set `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_VIEW_STATE_POSITION_VALID_BIT` but it **can** clear `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` and `XR_VIEW_STATE_POSITION_TRACKED_BIT` to indicate that the position is inferred or last-known in this way. When tracking is recovered, runtimes **should** snap the pose of other spaces back into position relative to the original origin of the **STAGE space**.

#### 7.1.4. Local Floor Reference Space

Local floor reference space, indicated by `XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR`, is closely related to the **LOCAL** reference space. It always aligns with the **LOCAL space**, and matches it in X and Z position. However, unlike the **LOCAL** space, the **LOCAL\_FLOOR** space has its Y axis origin on the runtime's best estimate of the floor level under the origin of the **LOCAL** space.

The location of the origin of the **LOCAL\_FLOOR** space **must** match the **LOCAL space** in the X and Z coordinates but not in the Y coordinate.

The orientation of the **LOCAL\_FLOOR** space **must** match the **LOCAL space**.

The runtime **must** establish the Y axis origin at its best estimate of the floor level under the origin of the **LOCAL space**, subject to requirements under the following conditions to match the floor level of the **STAGE space**.

If all of the following conditions are true, the Y axis origin of the **LOCAL\_FLOOR space** **must** match the Y axis origin of the **STAGE space**:

- the **STAGE space** is supported
- the location of the **LOCAL space** relative to the **STAGE** space has valid position (`XR_SPACE_LOCATION_POSITION_VALID_BIT` is set)
- bounds are available from `xrGetReferenceSpaceBoundsRect` for the **STAGE** space
- the position of the **LOCAL** space relative to the **STAGE** space is within the **STAGE** space XZ bounds

That is, if there is a stage with bounds, and if the local space and thus the local floor is logically within the stage, the local floor and the stage share the same floor level.

When the origin of the **LOCAL space** is changed in orientation or XZ position, the origin of the

`LOCAL_FLOOR` space **must** also change accordingly.

When a change in origin of the `LOCAL_FLOOR` space occurs, the runtime **must** queue the `XrEventDataReferenceSpaceChangePending` event, with the changed `LOCAL_FLOOR` space origin only taking effect for `xrLocateSpace` or `xrLocateViews` calls whose `XrTime` parameter is greater than or equal to the `XrEventDataReferenceSpaceChangePending::changeTime` in that event.

The `xrGetReferenceSpaceBoundsRect` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetReferenceSpaceBoundsRect(
    XrSession session,
    XrReferenceSpaceType referenceSpaceType,
    XrExtent2Df* bounds);
```

### Parameter Descriptions

- `session` is a handle to an `XrSession` previously created with `xrCreateSession`.
- `referenceSpaceType` is the reference space type whose bounds should be retrieved.
- `bounds` is the returned space extents.

XR systems **may** have limited real world spatial ranges in which users can freely move around while remaining tracked. Applications sometimes wish to query these boundaries and alter application behavior or content placement to ensure the user can complete the experience while remaining within the boundary. Applications **can** query this information using `xrGetReferenceSpaceBoundsRect`.

When called, `xrGetReferenceSpaceBoundsRect` **should** return the extents of a rectangle that is clear of obstacles down to the floor, allowing where the user can freely move while remaining tracked, if available for that reference space. The returned extent represents the dimensions of an axis-aligned bounding box where the `XrExtent2Df::width` and `XrExtent2Df::height` fields correspond to the X and Z axes of the provided space, with the extents centered at the origin of the space. Not all systems or spaces support boundaries. If a runtime is unable to provide bounds for a given space, `XR_SPACE_BOUNDS_UNAVAILABLE` **must** be returned and all fields of `bounds` **must** be set to 0.

The returned extents are expressed relative to the natural origin of the provided `XrReferenceSpaceType` and **must** not incorporate any origin offsets specified by the application during calls to `xrCreateReferenceSpace`.

The runtime **must** return `XR_ERROR_REFERENCE_SPACE_UNSUPPORTED` if the `XrReferenceSpaceType` passed in `referenceSpaceType` is not supported by this `session`.

When a runtime will begin operating with updated space bounds, the runtime **must** queue a

corresponding [XrEventDataReferenceSpaceChangePending](#) event.

### Valid Usage (Implicit)

- **session** **must** be a valid [XrSession](#) handle
- **referenceSpaceType** **must** be a valid [XrReferenceSpaceType](#) value
- **bounds** **must** be a pointer to an [XrExtent2Df](#) structure

### Return Codes

#### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)
- [XR\\_SPACE\\_BOUNDS\\_UNAVAILABLE](#)

#### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_REFERENCE\\_SPACE\\_UNSUPPORTED](#)

The [XrEventDataReferenceSpaceChangePending](#) event structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrEventDataReferenceSpaceChangePending {
    XrStructureType      type;
    const void*         next;
    XrSession            session;
    XrReferenceSpaceType referenceSpaceType;
    XrTime               changeTime;
    XrBool32             poseValid;
    XrPosef              poseInPreviousSpace;
} XrEventDataReferenceSpaceChangePending;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `session` is the `XrSession` for which the reference space is changing.
- `referenceSpaceType` is the `XrReferenceSpaceType` that is changing.
- `changeTime` is the target `XrTime` after which `xrLocateSpace` or `xrLocateViews` will return values that respect this change.
- `poseValid` is true if the runtime can determine the `poseInPreviousSpace` of the new space in the previous space before the change.
- `poseInPreviousSpace` is an `XrPosef` defining the position and orientation of the new reference space's natural origin within the natural reference frame of its previous space.

The `XrEventDataReferenceSpaceChangePending` event is sent to the application to notify it that the origin (and perhaps the bounds) of a reference space is changing. This may occur due to the user recentering the space explicitly, or the runtime otherwise switching to a different space definition.

The reference space change **must** only take effect for `xrLocateSpace` or `xrLocateViews` calls whose `XrTime` parameter is greater than or equal to the `changeTime` provided in that event. Runtimes **should** provide a `changeTime` to applications that allows for a deep render pipeline to present frames that are already in flight using the previous definition of the space. Runtimes **should** choose a `changeTime` that is midway between the `XrFrameState::predictedDisplayTime` of future frames to avoid threshold issues with applications that calculate future frame times using `XrFrameState::predictedDisplayPeriod`.

The `poseInPreviousSpace` provided here **must** only describe the change in the natural origin of the reference space and **must** not incorporate any origin offsets specified by the application during calls to `xrCreateReferenceSpace`. If the runtime does not know the location of the space's new origin relative to its previous origin, `poseValid` **must** be false, and the position and orientation of `poseInPreviousSpace` are undefined. .Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_EVENT_DATA_REFERENCE_SPACE_CHANGE_PENDING`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 7.2. Action Spaces

An `XrSpace` handle for a pose action is created using `xrCreateActionSpace`, by specifying the chosen pose action and a pose within the action's natural reference frame.

Runtimes support suggested pose action bindings to well-known user paths with `.../pose` subpaths if

they support tracking for that particular identifier.

Some example well-known pose action paths:

- `/user/hand/left/input/grip`
- `/user/hand/left/input/aim`
- `/user/hand/right/input/grip`
- `/user/hand/right/input/aim`

For definitions of these well-known pose device paths, see the discussion of [device input subpaths](#) in the Semantic Paths chapter.

### 7.2.1. Action Spaces Lifetime

`XrSpace` handles created for a pose action **must** be unlocatable unless the action set that contains the corresponding pose action was set as active via the most recent `xrSyncActions` call. If the underlying device that is active for the action changes, the device this space is tracking **must** only change to track the new device when `xrSyncActions` is called.

If `xrLocateSpace` is called with an unlocatable action space, the implementation **must** return no position or orientation and both `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` **must** be unset. If `XrSpaceVelocity` is also supplied, `XR_SPACE_VELOCITY_LINEAR_VALID_BIT` and `XR_SPACE_VELOCITY_ANGULAR_VALID_BIT` **must** be unset. If `xrLocateViews` is called with an unlocatable action space, the implementation **must** return no position or orientation and both `XR_VIEW_STATE_POSITION_VALID_BIT` and `XR_VIEW_STATE_ORIENTATION_VALID_BIT` **must** be unset.

## 7.3. Space Lifecycle

There are a small set of core APIs that allow applications to reason about reference spaces, action spaces, and their relative locations.

### 7.3.1. `xrEnumerateReferenceSpaces`

The `xrEnumerateReferenceSpaces` function is defined as:



```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateReferenceSpaces(
    XrSession session,
    uint32_t spaceCapacityInput,
    uint32_t* spaceCountOutput,
    XrReferenceSpaceType* spaces);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#) previously created with [xrCreateSession](#).
- `spaceCapacityInput` is the capacity of the `spaces` array, or 0 to indicate a request to retrieve the required capacity.
- `spaceCountOutput` is a pointer to the count of `spaces` written, or a pointer to the required capacity in the case that `spaceCapacityInput` is insufficient.
- `spaces` is a pointer to an application-allocated array that will be filled with the enumerant of each supported reference space. It **can** be `NULL` if `spaceCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `spaces` size.

Enumerates the set of reference space types that this runtime supports for a given session. Runtimes **must** always return identical buffer contents from this enumeration for the lifetime of the session.

If a session enumerates support for a given reference space type, calls to [xrCreateReferenceSpace](#) **must** succeed for that session, with any transient unavailability of poses expressed later during calls to [xrLocateSpace](#).

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `spaceCountOutput` **must** be a pointer to a `uint32_t` value
- If `spaceCapacityInput` is not 0, `spaces` **must** be a pointer to an array of `spaceCapacityInput` [XrReferenceSpaceType](#) values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

### 7.3.2. `xrCreateReferenceSpace`

The `xrCreateReferenceSpace` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrCreateReferenceSpace(
    XrSession session,
    const XrReferenceSpaceCreateInfo* createInfo,
    XrSpace* space);
```

### Parameter Descriptions

- `session` is a handle to an `XrSession` previously created with `xrCreateSession`.
- `createInfo` is the `XrReferenceSpaceCreateInfo` used to specify the space.
- `space` is the returned space handle.

Creates an `XrSpace` handle based on a chosen reference space. Application **can** provide an `XrPosef` to define the position and orientation of the new space's origin within the natural reference frame of the reference space.

Multiple `XrSpace` handles may exist simultaneously, up to some limit imposed by the runtime. The `XrSpace` handle **must** be eventually freed via the `xrDestroySpace` function.

The runtime **must** return `XR_ERROR_REFERENCE_SPACE_UNSUPPORTED` if the given reference space type is not supported by this `session`.

### Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrReferenceSpaceCreateInfo` structure
- `space` **must** be a pointer to an `XrSpace` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_REFERENCE_SPACE_UNSUPPORTED`
- `XR_ERROR_POSE_INVALID`

The `XrReferenceSpaceCreateInfo` structure is defined as:

```
typedef struct XrReferenceSpaceCreateInfo {
    XrStructureType      type;
    const void*          next;
    XrReferenceSpaceType referenceSpaceType;
    XrPosef              poseInReferenceSpace;
} XrReferenceSpaceCreateInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `referenceSpaceType` is the chosen [XrReferenceSpaceType](#).
- `poseInReferenceSpace` is an [XrPosef](#) defining the position and orientation of the new space's origin within the natural reference frame of the reference space.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_REFERENCE_SPACE_CREATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `referenceSpaceType` **must** be a valid [XrReferenceSpaceType](#) value

### 7.3.3. `xrCreateActionSpace`

The `xrCreateActionSpace` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrCreateActionSpace(
    XrSession session,
    const XrActionSpaceCreateInfo* createInfo,
    XrSpace* space);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to create the action space in.
- `createInfo` is the [XrActionSpaceCreateInfo](#) used to specify the space.
- `space` is the returned space handle.

Creates an [XrSpace](#) handle based on a chosen pose action. Application **can** provide an [XrPosef](#) to define the position and orientation of the new space's origin within the natural reference frame of the action space.

Multiple [XrSpace](#) handles may exist simultaneously, up to some limit imposed by the runtime. The

`XrSpace` handle must be eventually freed via the `xrDestroySpace` function or by destroying the parent `XrAction` handle.

The runtime **must** return `XR_ERROR_ACTION_TYPE_MISMATCH` if the action provided in `XrActionSpaceCreateInfo::action` is not of type `XR_ACTION_TYPE_POSE_INPUT`.

### Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrActionSpaceCreateInfo` structure
- `space` **must** be a pointer to an `XrSpace` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`

The `XrActionSpaceCreateInfo` structure is defined as:

```
typedef struct XrActionSpaceCreateInfo {
    XrStructureType    type;
    const void*        next;
    XrAction            action;
    XrPath              subactionPath;
    XrPosef             poseInActionSpace;
} XrActionSpaceCreateInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `action` is a handle to a pose [XrAction](#) previously created with [xrCreateAction](#).
- `subactionPath` is `XR_NULL_PATH` or an [XrPath](#) that was specified when the action was created. If `subactionPath` is a valid path not specified when the action was created the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`. If this parameter is set, the runtime **must** create a space that is relative to only that subaction's pose binding.
- `poseInActionSpace` is an [XrPosef](#) defining the position and orientation of the new space's origin within the natural reference frame of the pose action.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTION_SPACE_CREATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `action` **must** be a valid [XrAction](#) handle

### 7.3.4. xrDestroySpace

The [xrDestroySpace](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrDestroySpace(
    XrSpace                                space);
```

## Parameter Descriptions

- `space` is a handle to an `XrSpace` previously created by a function such as `xrCreateReferenceSpace`.

`XrSpace` handles are destroyed using `xrDestroySpace`. The runtime **may** still use this space if there are active dependencies (e.g, compositions in progress).

## Valid Usage (Implicit)

- `space` **must** be a valid `XrSpace` handle

## Thread Safety

- Access to `space`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_HANDLE_INVALID`

## 7.4. Locating Spaces

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. If an application wants to know the velocity of the space's origin, it **can** chain an `XrSpaceVelocity` structure to the `next` pointer of the `XrSpaceLocation` structure when calling the `xrLocateSpace` function. Applications **should** inspect the output `XrSpaceLocationFlagBits` and `XrSpaceVelocityFlagBits` to determine the validity and tracking status of the components of the location.

### 7.4.1. `xrLocateSpace`

`xrLocateSpace` provides the physical location of a space in a base space at a specified time, if currently known by the runtime.

```
// Provided by XR_VERSION_1_0
XrResult xrLocateSpace(
    XrSpace          space,
    XrSpace          baseSpace,
    XrTime           time,
    XrSpaceLocation* location);
```

## Parameter Descriptions

- **space** identifies the target space to locate.
- **baseSpace** identifies the underlying space in which to locate **space**.
- **time** is the time for which the location should be provided.
- **location** provides the location of **space** in **baseSpace**.

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](#). For values of **time** outside this range, **xrLocateSpace** **may** return a location with no position and **XR\_SPACE\_LOCATION\_POSITION\_VALID\_BIT** unset.

Some devices improve their understanding of the world as the device is used. The location returned by **xrLocateSpace** for a given **space**, **baseSpace** and **time** **may** change over time, even for spaces that track static objects, as one or both spaces adjust their origins.

During tracking loss of **space** relative to **baseSpace**, runtimes **should** continue to provide inferred or last-known **XrPosef::position** and **XrPosef::orientation** values. These inferred poses can, for example, be based on neck model updates, inertial dead reckoning, or a last-known position, so long as it is still reasonable for the application to use that pose. While a runtime is providing position data, it **must** continue to set **XR\_SPACE\_LOCATION\_POSITION\_VALID\_BIT** but it **can** clear **XR\_SPACE\_LOCATION\_POSITION\_TRACKED\_BIT** to indicate that the position is inferred or last-known in this way.

If the runtime has not yet observed even a last-known pose for how to locate **space** in **baseSpace** (e.g. one space is an action space bound to a motion controller that has not yet been detected, or the two spaces are in disconnected fragments of the runtime's tracked volume), the runtime **should** return a location with no position and **XR\_SPACE\_LOCATION\_POSITION\_VALID\_BIT** unset.

The runtime **must** return a location with both **XR\_SPACE\_LOCATION\_POSITION\_VALID\_BIT** and **XR\_SPACE\_LOCATION\_POSITION\_TRACKED\_BIT** set when locating **space** and **baseSpace** if both spaces were



created relative to the same entity (e.g. two action spaces for the same action), even if the entity is currently untracked. The location in this case is the difference in the two spaces' application-specified transforms relative to that common entity.

During tracking loss, the runtime **should** return a location with `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` set and `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` and `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` unset for spaces tracking two static entities in the world when their relative pose is known to the runtime. This enables applications to continue to make use of the runtime's latest knowledge of the world.

If an `XrSpaceVelocity` structure is chained to the `XrSpaceLocation::next` pointer, and the velocity is observed or can be calculated by the runtime, the runtime **must** fill in the linear velocity of the origin of space within the reference frame of `baseSpace` and set the `XR_SPACE_VELOCITY_LINEAR_VALID_BIT`. Similarly, if an `XrSpaceVelocity` structure is chained to the `XrSpaceLocation::next` pointer, and the angular velocity is observed or can be calculated by the runtime, the runtime **must** fill in the angular velocity of the origin of space within the reference frame of `baseSpace` and set the `XR_SPACE_VELOCITY_ANGULAR_VALID_BIT`.

The following example code shows how an application can get both the location and velocity of a space within a base space using the `xrLocateSpace` function by chaining an `XrSpaceVelocity` to the `next` pointer of `XrSpaceLocation` and calling `xrLocateSpace`.

```
XrSpace space;          // previously initialized
XrSpace baseSpace;     // previously initialized
XrTime time;           // previously initialized

XrSpaceVelocity velocity {XR_TYPE_SPACE_VELOCITY};
XrSpaceLocation location {XR_TYPE_SPACE_LOCATION, &velocity};
xrLocateSpace(space, baseSpace, time, &location);
```

### Valid Usage (Implicit)

- `space` **must** be a valid `XrSpace` handle
- `baseSpace` **must** be a valid `XrSpace` handle
- `location` **must** be a pointer to an `XrSpaceLocation` structure
- Both of `baseSpace` and `space` **must** have been created, allocated, or retrieved from the same `XrSession`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrSpaceLocation` structure is defined as:

```
typedef struct XrSpaceLocation {
    XrStructureType    type;
    void*              next;
    XrSpaceLocationFlags locationFlags;
    XrPosef            pose;
} XrSpaceLocation;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain, such as `XrSpaceVelocity`.
- `locationFlags` is a bitfield, with bit masks defined in `XrSpaceLocationFlagBits`, to indicate which members contain valid data. If none of the bits are set, no other fields in this structure **should** be considered to be valid or meaningful.
- `pose` is an `XrPosef` defining the position and orientation of the origin of `xrLocateSpace::space` within the reference frame of `xrLocateSpace::baseSpace`.

## Valid Usage (Implicit)

- **type** must be `XR_TYPE_SPACE_LOCATION`
- **next** must be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrEyeGazeSampleTimeEXT](#), [XrSpaceVelocity](#)
- **locationFlags** must be `0` or a valid combination of [XrSpaceLocationFlagBits](#) values

The `XrSpaceLocation::locationFlags` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in [XrSpaceLocationFlagBits](#).

```
typedef XrFlags64 XrSpaceLocationFlags;
```

Valid bits for [XrSpaceLocationFlags](#) are defined by [XrSpaceLocationFlagBits](#), which is specified as:

```
// Flag bits for XrSpaceLocationFlags
static const XrSpaceLocationFlags XR_SPACE_LOCATION_ORIENTATION_VALID_BIT = 0x00000001;
static const XrSpaceLocationFlags XR_SPACE_LOCATION_POSITION_VALID_BIT = 0x00000002;
static const XrSpaceLocationFlags XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT = 0x00000004;
static const XrSpaceLocationFlags XR_SPACE_LOCATION_POSITION_TRACKED_BIT = 0x00000008;
```

The flag bits have the following meanings:

## Flag Descriptions

- `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` indicates that the `pose` field's `orientation` field contains valid data. For a space location tracking a device with its own inertial tracking, `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` **should** remain set when this bit is set. Applications **must** not read the `pose` field's `orientation` if this flag is unset.
- `XR_SPACE_LOCATION_POSITION_VALID_BIT` indicates that the `pose` field's `position` field contains valid data. When a space location loses tracking, runtimes **should** continue to provide valid but untracked `position` values that are inferred or last-known, so long as it's still meaningful for the application to use that position, clearing `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` until positional tracking is recovered. Applications **must** not read the `pose` field's `position` if this flag is unset.
- `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` indicates that the `pose` field's `orientation` field represents an actively tracked orientation. For a space location tracking a device with its own inertial tracking, this bit **should** remain set when `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` is set. For a space location tracking an object whose orientation is no longer known during tracking loss (e.g. an observed QR code), runtimes **should** continue to provide valid but untracked `orientation` values, so long as it's still meaningful for the application to use that orientation.
- `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` indicates that the `pose` field's `position` field represents an actively tracked position. When a space location loses tracking, runtimes **should** continue to provide valid but untracked `position` values that are inferred or last-known, e.g. based on neck model updates, inertial dead reckoning, or a last-known position, so long as it's still meaningful for the application to use that position.

The `XrSpaceVelocity` structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrSpaceVelocity {
    XrStructureType    type;
    void*              next;
    XrSpaceVelocityFlags velocityFlags;
    XrVector3f         linearVelocity;
    XrVector3f         angularVelocity;
} XrSpaceVelocity;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `velocityFlags` is a bitfield, with bit masks defined in [XrSpaceVelocityFlagBits](#), to indicate which members contain valid data. If none of the bits are set, no other fields in this structure **should** be considered to be valid or meaningful.
- `linearVelocity` is the relative linear velocity of the origin of [xrLocateSpace::space](#) with respect to and expressed in the reference frame of [xrLocateSpace::baseSpace](#), in units of meters per second.
- `angularVelocity` is the relative angular velocity of [xrLocateSpace::space](#) with respect to [xrLocateSpace::baseSpace](#). The vector's direction is expressed in the reference frame of [xrLocateSpace::baseSpace](#) and is parallel to the rotational axis of [xrLocateSpace::space](#). The vector's magnitude is the relative angular speed of [xrLocateSpace::space](#) in radians per second. The vector follows the right-hand rule for torque/rotation.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SPACE_VELOCITY`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `velocityFlags` **must** be `0` or a valid combination of [XrSpaceVelocityFlagBits](#) values

The [XrSpaceVelocity::velocityFlags](#) member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in [XrSpaceVelocityFlagBits](#).

```
typedef XrFlags64 XrSpaceVelocityFlags;
```

Valid bits for [XrSpaceVelocityFlags](#) are defined by [XrSpaceVelocityFlagBits](#), which is specified as:

```
// Flag bits for XrSpaceVelocityFlags
static const XrSpaceVelocityFlags XR_SPACE_VELOCITY_LINEAR_VALID_BIT = 0x00000001;
static const XrSpaceVelocityFlags XR_SPACE_VELOCITY_ANGULAR_VALID_BIT = 0x00000002;
```

The flag bits have the following meanings:

## Flag Descriptions

- `XR_SPACE_VELOCITY_LINEAR_VALID_BIT` — Indicates that the `linearVelocity` member contains valid data. Applications **must** not read the `linearVelocity` field if this flag is unset.
- `XR_SPACE_VELOCITY_ANGULAR_VALID_BIT` — Indicates that the `angularVelocity` member contains valid data. Applications **must** not read the `angularVelocity` field if this flag is unset.

### 7.4.2. Locate spaces

Applications **can** use `xrLocateSpaces` function to locate an array of spaces.

The `xrLocateSpaces` function is defined as:

```
// Provided by XR_VERSION_1_1
XrResult xrLocateSpaces(
    XrSession session,
    const XrSpacesLocateInfo* locateInfo,
    XrSpaceLocations* spaceLocations);
```

## Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `locateInfo` is a pointer to an `XrSpacesLocateInfo` that provides the input information to locate spaces.
- `spaceLocations` is a pointer to an `XrSpaceLocations` for the runtime to return the locations of the specified spaces in the base space.

`xrLocateSpaces` provides the physical location of one or more spaces in a base space at a specified time, if currently known by the runtime.

The `XrSpacesLocateInfo::time`, the `XrSpacesLocateInfo::baseSpace`, and each space in `XrSpacesLocateInfo::spaces`, in the `locateInfo` parameter, all follow the same specifics as the corresponding inputs to the `xrLocateSpace` function.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `locateInfo` **must** be a pointer to a valid `XrSpacesLocateInfo` structure
- `spaceLocations` **must** be a pointer to an `XrSpaceLocations` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_TIME_INVALID`

The `XrSpacesLocateInfo` structure is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrSpacesLocateInfo {
    XrStructureType    type;
    const void*        next;
    XrSpace             baseSpace;
    XrTime              time;
    uint32_t           spaceCount;
    const XrSpace*     spaces;
} XrSpacesLocateInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `baseSpace` identifies the underlying space in which to locate `spaces`.
- `time` is the time for which the location is requested.
- `spaceCount` is a `uint32_t` specifying the count of elements in the `spaces` array.
- `spaces` is an array of valid `XrSpace` handles to be located.

The `time`, the `baseSpace`, and each space in `spaces` all follow the same specifics as the corresponding inputs to the `xrLocateSpace` function.

The `baseSpace` and all of the `XrSpace` handles in the `spaces` array **must** be valid and share the same parent `XrSession`.

If the `time` is invalid, the `xrLocateSpaces` **must** return `XR_ERROR_TIME_INVALID`.

The `spaceCount` **must** be a positive number, i.e. the array `spaces` **must** not be empty. Otherwise, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SPACES_LOCATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `baseSpace` **must** be a valid `XrSpace` handle
- `spaces` **must** be a pointer to an array of `spaceCount` valid `XrSpace` handles
- The `spaceCount` parameter **must** be greater than `0`
- Both of `baseSpace` and the elements of `spaces` **must** have been created, allocated, or retrieved from the same `XrSession`

The `XrSpaceLocations` structure is defined as:



```
// Provided by XR_VERSION_1_1
typedef struct XrSpaceLocations {
    XrStructureType    type;
    void*              next;
    uint32_t           locationCount;
    XrSpaceLocationData* locations;
} XrSpaceLocations;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain, such as `XrSpaceVelocities`.
- `locationCount` is a `uint32_t` specifying the count of elements in the `locations` array.
- `locations` is an array of `XrSpaceLocations` for the runtime to populate with the locations of the specified spaces in the `XrSpacesLocateInfo::baseSpace` at the specified `XrSpacesLocateInfo::time`.

The `XrSpaceLocations` structure contains an array of space locations in the member `locations`, to be used as output for `xrLocateSpaces`. The application **must** allocate this array to be populated with the function output. The `locationCount` value **must** be the same as `XrSpacesLocateInfo::spaceCount`, otherwise, the `xrLocateSpaces` function **must** return `XR_ERROR_VALIDATION_FAILURE`.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SPACE_LOCATIONS`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`. See also: `XrSpaceVelocities`
- `locations` **must** be a pointer to an array of `locationCount` `XrSpaceLocationData` structures
- The `locationCount` parameter **must** be greater than `0`

The `XrSpaceLocationData` structure is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrSpaceLocationData {
    XrSpaceLocationFlags locationFlags;
    XrPosef               pose;
} XrSpaceLocationData;
```

## Member Descriptions

- `locationFlags` is a bitfield, with bit masks defined in `XrSpaceLocationFlagBits`. It behaves the same as `XrSpaceLocation::locationFlags`.
- `pose` is an `XrPosef` that behaves the same as `XrSpaceLocation::pose`.

This is a single element of the array in `XrSpaceLocations::locations`, and is used to return the pose and location flags for a single space with respect to the specified base space from a call to `xrLocateSpaces`. It does not accept chained structures to allow for easier use in dynamically allocated container datatypes. Chained structures are possible with the `XrSpaceLocations` that describes an array of these elements.

### 7.4.3. Locate space velocities

Applications **can** request the velocities of spaces by chaining the `XrSpaceVelocities` structure to the next pointer of `XrSpaceLocations` when calling `xrLocateSpaces`.

The `XrSpaceVelocities` structure is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrSpaceVelocities {
    XrStructureType    type;
    void*              next;
    uint32_t           velocityCount;
    XrSpaceVelocityData* velocities;
} XrSpaceVelocities;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `velocityCount` is a `uint32_t` specifying the count of elements in the `velocities` array.
- `velocities` is an array of `XrSpaceVelocityData` for the runtime to populate with the velocities of the specified spaces in the `XrSpacesLocateInfo::baseSpace` at the specified `XrSpacesLocateInfo::time`.

The `velocities` member contains an array of space velocities in the member `velocities`, to be used as output for `xrLocateSpaces`. The application **must** allocate this array to be populated with the function output. The `velocityCount` value **must** be the same as `XrSpacesLocateInfo::spaceCount`, otherwise, the

`xrLocateSpaces` function **must** return `XR_ERROR_VALIDATION_FAILURE`.

### Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SPACE_VELOCITIES`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `velocities` **must** be a pointer to an array of `velocityCount` `XrSpaceVelocityData` structures
- The `velocityCount` parameter **must** be greater than `0`

The `XrSpaceVelocityData` structure is defined as:

```
// Provided by XR_VERSION_1_1
typedef struct XrSpaceVelocityData {
    XrSpaceVelocityFlags    velocityFlags;
    XrVector3f              linearVelocity;
    XrVector3f              angularVelocity;
} XrSpaceVelocityData;
```

### Member Descriptions

- `velocityFlags` is a bitfield, with bit values defined in `XrSpaceVelocityFlagBits`. It behaves the same as `XrSpaceVelocity::velocityFlags`.
- `linearVelocity` is an `XrVector3f`. It behaves the same as `XrSpaceVelocity::linearVelocity`.
- `angularVelocity` is an `XrVector3f`. It behaves the same as `XrSpaceVelocity::angularVelocity`.

This is a single element of the array in `XrSpaceVelocities::velocities`, and is used to return the linear and angular velocity and velocity flags for a single space with respect to the specified base space from a call to `xrLocateSpaces`. It does not accept chained structures to allow for easier use in dynamically allocated container datatypes.

#### 7.4.4. Example code for `xrLocateSpaces`

The following example code shows how an application retrieves both the location and velocity of one or more spaces in a base space at a given time using the `xrLocateSpaces` function.

```
XrInstance instance; // previously initialized
XrSession session; // previously initialized
XrSpace baseSpace; // previously initialized
```

```

std::vector<XrSpace> spacesToLocate; // previously initialized

// Prepare output buffers to receive data and get reused in frame loop.
std::vector<XrSpaceLocationData> locationBuffer(spacesToLocate.size());
std::vector<XrSpaceVelocityData> velocityBuffer(spacesToLocate.size());

// Get function pointer for xrLocateSpaces.
PFN_xrLocateSpaces xrLocateSpaces;
CHK_XR(xrGetInstanceProcAddr(instance, "xrLocateSpaces",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &xrLocateSpaces)));

// application frame loop
while (1) {
    // Typically the time is the predicted display time returned from xrWaitFrame.
    XrTime displayTime; // previously initialized.

    XrSpacesLocateInfo locateInfo{XR_TYPE_SPACES_LOCATE_INFO};
    locateInfo.baseSpace = baseSpace;
    locateInfo.time = displayTime;
    locateInfo.spaceCount = (uint32_t)spacesToLocate.size();
    locateInfo.spaces = spacesToLocate.data();

    XrSpaceLocations locations{XR_TYPE_SPACE_LOCATIONS};
    locations.locationCount = (uint32_t)locationBuffer.size();
    locations.locations = locationBuffer.data();

    XrSpaceVelocities velocities{XR_TYPE_SPACE_VELOCITIES};
    velocities.velocityCount = (uint32_t)velocityBuffer.size();
    velocities.velocities = velocityBuffer.data();

    locations.next = &velocities;
    CHK_XR(xrLocateSpaces(session, &locateInfo, &locations));

    for (uint32_t i = 0; i < spacesToLocate.size(); i++) {
        const auto positionAndOrientationTracked =
            XR_SPACE_LOCATION_POSITION_TRACKED_BIT |
XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT;
        const auto orientationOnlyTracked = XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT;

        if ((locationBuffer[i].locationFlags & positionAndOrientationTracked) ==
positionAndOrientationTracked) {
            // if the location is 6dof tracked
            do_something(locationBuffer[i].pose.position);
            do_something(locationBuffer[i].pose.orientation);

            const auto velocityValidBits =
                XR_SPACE_VELOCITY_LINEAR_VALID_BIT | XR_SPACE_VELOCITY_ANGULAR_VALID_BIT;

```

```

        if ((velocityBuffer[i].velocityFlags & velocityValidBits) ==
velocityValidBits) {
            do_something(velocityBuffer[i].linearVelocity);
            do_something(velocityBuffer[i].angularVelocity);
        }
    }
    else if ((locationBuffer[i].locationFlags & orientationOnlyTracked) ==
orientationOnlyTracked) {
        // if the location is 3dof tracked
        do_something(locationBuffer[i].pose.orientation);

        if ((velocityBuffer[i].velocityFlags & XR_SPACE_VELOCITY_ANGULAR_VALID_BIT)
== XR_SPACE_VELOCITY_ANGULAR_VALID_BIT) {
            do_something(velocityBuffer[i].angularVelocity);
        }
    }
}
}
}

```

# Chapter 8. View Configurations

A **view configuration** is a semantically meaningful set of one or more views for which an application can render images. A **primary view configuration** is a view configuration intended to be presented to the viewer interacting with the XR application. This distinction allows the later addition of additional views, for example views which are intended for spectators.

A typical head-mounted VR system has a view configuration with two views, while a typical phone-based AR system has a view configuration with a single view. A simple multi-wall projection-based (CAVE-like) VR system may have a view configuration with at least one view for each display surface (wall, floor, ceiling) in the room.

For any supported form factor, a system will support one or more primary view configurations. Supporting more than one primary view configuration can be useful if a system supports a special view configuration optimized for the hardware but also supports a more broadly used view configuration as a compatibility fallback.

View configurations are identified with an [XrViewConfigurationType](#).

## 8.1. Primary View Configurations

```
typedef enum XrViewConfigurationType {
    XR_VIEW_CONFIGURATION_TYPE_PRIMARY_MONO = 1,
    XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO = 2,
    // Provided by XR_VERSION_1_1
    XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET = 1000037000,
    // Provided by XR_MSFT_first_person_observer
    XR_VIEW_CONFIGURATION_TYPE_SECONDARY_MONO_FIRST_PERSON_OBSERVER_MSFT = 1000054000,
    // Provided by XR_VARJO_quad_views
    XR_VIEW_CONFIGURATION_TYPE_PRIMARY_QUAD_VARJO =
XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET,
    XR_VIEW_CONFIGURATION_TYPE_MAX_ENUM = 0x7FFFFFFF
} XrViewConfigurationType;
```

The application selects its primary view configuration type when calling [xrBeginSession](#), and that configuration remains constant for the lifetime of the session, until [xrEndSession](#) is called.

The number of views and the semantic meaning of each view index within a given view configuration is well-defined, specified below for all core view configurations. The predefined primary view configuration types are:

## Enumerant Descriptions

- `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_MONO`. One view representing the form factor's one primary display. For example, an AR phone's screen. This configuration requires one element in `XrViewConfigurationProperties` and one projection in each `XrCompositionLayerProjection` layer.
- `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO`. Two views representing the form factor's two primary displays, which map to a left-eye and right-eye view. This configuration requires two views in `XrViewConfigurationProperties` and two views in each `XrCompositionLayerProjection` layer. View index 0 **must** represent the left eye and view index 1 **must** represent the right eye.
- `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET`. Four views representing the form factor's primary stereo displays. This view configuration type represents a hardware independent way of providing foveated rendering. The view configuration adds two foveated inset views for the left and right eye separately to the already defined two views specified in the `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO` view configuration. View index 0 **must** represent the left eye and view index 1 **must** represent the right eye as specified in `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO` view configuration, and view index 2 **must** represent the left eye inset view and view index 3 **must** represent the right eye inset view. The new inset view 2 and view 3 **must**, after applying the pose and FoV projection to same plane, be contained within view 0 and 1 respectively. The inset views **may** have a higher resolution with respect to the same field of view as the corresponding wide FoV view for each eye. The runtime **may** blend between the views at the edges, so the application **must** not omit the inner field of view from being rendered in the outer view. The `fov` returned by `xrLocateViews` for each inset view relative to the corresponding outer stereo view **may** change at run-time, the `pose` for inset view and stereo view for each eye respectively **must** have the same values.



The benefits of the `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET` view configuration type can be demonstrated by looking at the rendered pixel count. For example, a Varjo Aero requires a pair of stereo views rendered at 4148 x 3556 (14.7 million pixels) to achieve a pixel density of 35 pixels per degree. By using four views, with an eye-tracked foveated inset covering about 1/9th of the full FoV and rendered with the same 35 pixels per degree and while the remaining views are dropped to 14 pixels per degree, the resolution of the inset is 1076 x 1076 (1.1 million pixels) and the resolution of the stereo views is 1660 x 1420 (2.3 million pixels). The total pixel count is 75% less with `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET` over the `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO` view configuration type.

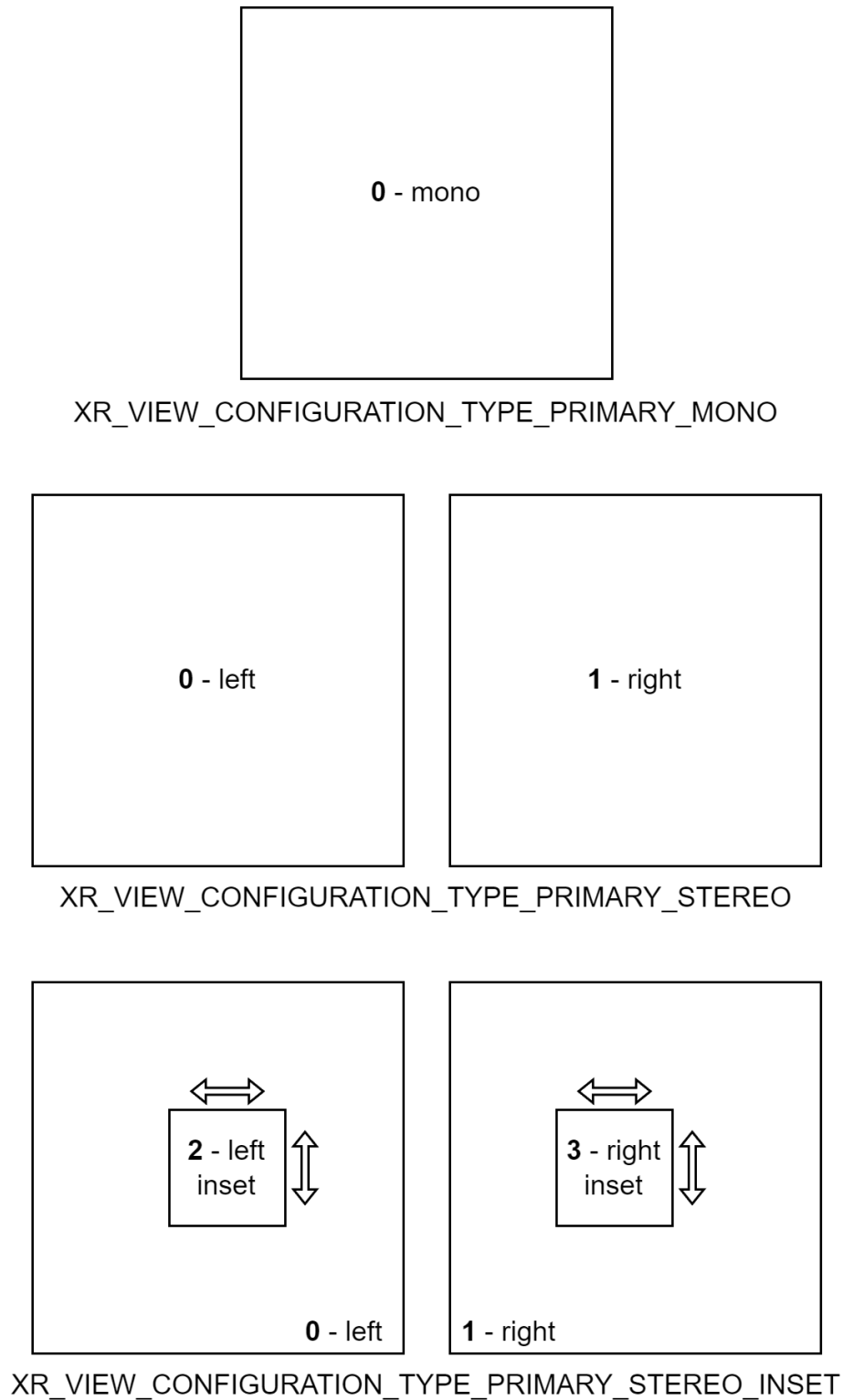


Figure 4. View configurations. The numbers in the figure is the view indices of the specific view.

## 8.2. View Configuration API

First an application needs to select which primary view configuration it wants to use. If it supports multiple configurations, an application **can** call [xrEnumerateViewConfigurations](#) before creating an



[XrSession](#) to get a list of the view configuration types supported for a given system.

The application **can** then call [xrGetViewConfigurationProperties](#) and [xrEnumerateViewConfigurationViews](#) to get detailed information about each view configuration type and its individual views.

### 8.2.1. xrEnumerateViewConfigurations

The [xrEnumerateViewConfigurations](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateViewConfigurations(
    XrInstance                instance,
    XrSystemId               systemId,
    uint32_t                 viewConfigurationTypeCapacityInput,
    uint32_t*                viewConfigurationTypeCountOutput,
    XrViewConfigurationType* viewConfigurationTypes);
```

#### Parameter Descriptions

- **instance** is the instance from which **systemId** was retrieved.
- **systemId** is the [XrSystemId](#) whose view configurations will be enumerated.
- **viewConfigurationTypeCapacityInput** is the capacity of the **viewConfigurationTypes** array, or 0 to indicate a request to retrieve the required capacity.
- **viewConfigurationTypeCountOutput** is a pointer to the count of **viewConfigurationTypes** written, or a pointer to the required capacity in the case that **viewConfigurationTypeCapacityInput** is insufficient.
- **viewConfigurationTypes** is a pointer to an array of [XrViewConfigurationType](#) values, but **can** be NULL if **viewConfigurationTypeCapacityInput** is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required **viewConfigurationTypes** size.

[xrEnumerateViewConfigurations](#) enumerates the view configuration types supported by the [XrSystemId](#). The supported set for that system **must** not change during the lifetime of its [XrInstance](#). The returned list of primary view configurations **should** be in order from what the runtime considered highest to lowest user preference. Thus the first enumerated view configuration type **should** be the one the runtime prefers the application to use if possible.

Runtimes **must** always return identical buffer contents from this enumeration for the given **systemId** and for the lifetime of the instance.

## Valid Usage (Implicit)

- `instance` **must** be a valid [XrInstance](#) handle
- `viewConfigurationTypeCountOutput` **must** be a pointer to a `uint32_t` value
- If `viewConfigurationTypeCapacityInput` is not `0`, `viewConfigurationTypes` **must** be a pointer to an array of `viewConfigurationTypeCapacityInput` [XrViewConfigurationType](#) values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SYSTEM_INVALID`

## 8.2.2. `xrGetViewConfigurationProperties`

The `xrGetViewConfigurationProperties` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetViewConfigurationProperties(
    XrInstance           instance,
    XrSystemId          systemId,
    XrViewConfigurationType viewConfigurationType,
    XrViewConfigurationProperties* configurationProperties);
```

## Parameter Descriptions

- `instance` is the instance from which `systemId` was retrieved.
- `systemId` is the `XrSystemId` whose view configuration is being queried.
- `viewConfigurationType` is the `XrViewConfigurationType` of the configuration to get.
- `configurationProperties` is a pointer to view configuration properties to return.

`xrGetViewConfigurationProperties` queries properties of an individual view configuration. Applications **must** use one of the supported view configuration types returned by `xrEnumerateViewConfigurations`. If `viewConfigurationType` is not supported by this `XrInstance` the runtime **must** return `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `viewConfigurationType` **must** be a valid `XrViewConfigurationType` value
- `configurationProperties` **must** be a pointer to an `XrViewConfigurationProperties` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`
- `XR_ERROR_SYSTEM_INVALID`

### 8.2.3. `XrViewConfigurationProperties`

The `XrViewConfigurationProperties` structure is defined as:

```
typedef struct XrViewConfigurationProperties {
    XrStructureType      type;
    void*                next;
    XrViewConfigurationType viewConfigurationType;
    XrBool32             fovMutable;
} XrViewConfigurationProperties;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `viewConfigurationType` is the [XrViewConfigurationType](#) of the configuration.
- `fovMutable` indicates if the view field of view can be modified by the application.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_VIEW_CONFIGURATION_PROPERTIES`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `viewConfigurationType` **must** be a valid [XrViewConfigurationType](#) value

### 8.2.4. xrEnumerateViewConfigurationViews

The [xrEnumerateViewConfigurationViews](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateViewConfigurationViews(
    XrInstance          instance,
    XrSystemId         systemId,
    XrViewConfigurationType viewConfigurationType,
    uint32_t            viewCapacityInput,
    uint32_t*           viewCountOutput,
    XrViewConfigurationView* views);
```

## Parameter Descriptions

- `instance` is the instance from which `systemId` was retrieved.
- `systemId` is the `XrSystemId` whose view configuration is being queried.
- `viewConfigurationType` is the `XrViewConfigurationType` of the configuration to get.
- `viewCapacityInput` is the capacity of the `views` array, or 0 to indicate a request to retrieve the required capacity.
- `viewCountOutput` is a pointer to the count of `views` written, or a pointer to the required capacity in the case that `viewCapacityInput` is 0.
- `views` is a pointer to an array of `XrViewConfigurationView` values, but **can** be `NULL` if `viewCapacityInput` is 0.

Each `XrViewConfigurationType` defines the number of views associated with it. Applications can query more details of each view element using `xrEnumerateViewConfigurationViews`. If the supplied `viewConfigurationType` is not supported by this `XrInstance` and `XrSystemId`, the runtime **must** return `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`.

Runtimes **must** always return identical buffer contents from this enumeration for the given `systemId` and `viewConfigurationType` for the lifetime of the instance.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `viewConfigurationType` **must** be a valid `XrViewConfigurationType` value
- `viewCountOutput` **must** be a pointer to a `uint32_t` value
- If `viewCapacityInput` is not 0, `views` **must** be a pointer to an array of `viewCapacityInput` `XrViewConfigurationView` structures

## Return Codes

### Success

- XR\_SUCCESS

### Failure

- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_VIEW\_CONFIGURATION\_TYPE\_UNSUPPORTED
- XR\_ERROR\_SYSTEM\_INVALID

### 8.2.5. XrViewConfigurationView

Each [XrViewConfigurationView](#) specifies properties related to rendering of an individual view within a view configuration.

The [XrViewConfigurationView](#) structure is defined as:

```
typedef struct XrViewConfigurationView {
    XrStructureType    type;
    void*              next;
    uint32_t           recommendedImageRectWidth;
    uint32_t           maxImageRectWidth;
    uint32_t           recommendedImageRectHeight;
    uint32_t           maxImageRectHeight;
    uint32_t           recommendedSwapchainSampleCount;
    uint32_t           maxSwapchainSampleCount;
} XrViewConfigurationView;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `recommendedImageRectWidth` is the optimal width of [XrSwapchainSubImage::imageRect](#) to use when rendering this view into a swapchain.
- `maxImageRectWidth` is the maximum width of [XrSwapchainSubImage::imageRect](#) supported when rendering this view into a swapchain.
- `recommendedImageRectHeight` is the optimal height of [XrSwapchainSubImage::imageRect](#) to use when rendering this view into a swapchain.
- `maxImageRectHeight` is the maximum height of [XrSwapchainSubImage::imageRect](#) supported when rendering this view into a swapchain.
- `recommendedSwapchainSampleCount` is the recommended number of sub-data element samples to create for each swapchain image that will be rendered into for this view.
- `maxSwapchainSampleCount` is the maximum number of sub-data element samples supported for swapchain images that will be rendered into for this view.

See [XrSwapchainSubImage](#) for more information about [XrSwapchainSubImage::imageRect](#) values, and [XrSwapchainCreateInfo](#) for more information about creating swapchains appropriately sized to support those [XrSwapchainSubImage::imageRect](#) values.

The array of [XrViewConfigurationView](#) returned by the runtime **must** adhere to the rules defined in [XrViewConfigurationType](#), such as the count and association to the left and right eyes.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_VIEW_CONFIGURATION_VIEW`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrFoveatedViewConfigurationViewVARJO](#), [XrViewConfigurationDepthRangeEXT](#), [XrViewConfigurationViewFovEPIC](#)

## 8.3. Example View Configuration Code

```
XrInstance instance; // previously initialized
XrSystemId system;  // previously initialized
XrSession session;  // previously initialized
XrSpace sceneSpace; // previously initialized
```

```

// Enumerate the view configurations paths.
uint32_t configurationCount;
CHK_XR(xrEnumerateViewConfigurations(instance, system, 0, &configurationCount, nullptr));

std::vector<XrViewConfigurationType> configurationTypes(configurationCount);
CHK_XR(xrEnumerateViewConfigurations(instance, system, configurationCount,
&configurationCount, configurationTypes.data()));

bool configFound = false;
XrViewConfigurationType viewConfig = XR_VIEW_CONFIGURATION_TYPE_MAX_ENUM;
for(uint32_t i = 0; i < configurationCount; ++i)
{
    if (configurationTypes[i] == XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO)
    {
        configFound = true;
        viewConfig = configurationTypes[i];
        break; // Pick the first supported, i.e. preferred, view configuration.
    }
}

if (!configFound)
    return; // Cannot support any view configuration of this system.

// Get detailed information of each view element.
uint32_t viewCount;
CHK_XR(xrEnumerateViewConfigurationViews(instance, system,
    viewConfig,
    0,
    &viewCount,
    nullptr));

std::vector<XrViewConfigurationView> configViews(viewCount,
{XR_TYPE_VIEW_CONFIGURATION_VIEW});
CHK_XR(xrEnumerateViewConfigurationViews(instance, system,
    viewConfig,
    viewCount,
    &viewCount,
    configViews.data()));

// Set the primary view configuration for the session.
XrSessionBeginInfo beginInfo = {XR_TYPE_SESSION_BEGIN_INFO};
beginInfo.primaryViewConfigurationType = viewConfig;
CHK_XR(xrBeginSession(session, &beginInfo));

// Allocate a buffer according to viewCount.
std::vector<XrView> views(viewCount, {XR_TYPE_VIEW});

// Run a per-frame loop.

```



```

while (!quit)
{
    // Wait for a new frame.
    XrFrameWaitInfo frameWaitInfo{XR_TYPE_FRAME_WAIT_INFO};
    XrFrameState frameState{XR_TYPE_FRAME_STATE};
    CHK_XR(xrWaitFrame(session, &frameWaitInfo, &frameState));

    // Begin frame immediately before GPU work
    XrFrameBeginInfo frameBeginInfo { XR_TYPE_FRAME_BEGIN_INFO };
    CHK_XR(xrBeginFrame(session, &frameBeginInfo));

    std::vector<XrCompositionLayerBaseHeader*> layers;
    XrCompositionLayerProjectionView projViews[2] = { /*...*/ };
    XrCompositionLayerProjection layerProj{ XR_TYPE_COMPOSITION_LAYER_PROJECTION};

    if (frameState.shouldRender) {
        XrViewLocateInfo viewLocateInfo{XR_TYPE_VIEW_LOCATE_INFO};
        viewLocateInfo.viewConfigurationType = viewConfig;
        viewLocateInfo.displayTime = frameState.predictedDisplayTime;
        viewLocateInfo.space = sceneSpace;

        XrViewState viewState{XR_TYPE_VIEW_STATE};
        XrView views[2] = { {XR_TYPE_VIEW}, {XR_TYPE_VIEW}};
        uint32_t viewCountOutput;
        CHK_XR(xrLocateViews(session, &viewLocateInfo, &viewState, configViews.size(),
&viewCountOutput, views));

        // ...
        // Use viewState and frameState for scene render, and fill in projViews[2]
        // ...

        // Assemble composition layers structure
        layerProj.layerFlags = XR_COMPOSITION_LAYER_BLEND_TEXTURE_SOURCE_ALPHA_BIT;
        layerProj.space = sceneSpace;
        layerProj.viewCount = 2;
        layerProj.views = projViews;
        layers.push_back(reinterpret_cast<XrCompositionLayerBaseHeader*>(&layerProj));
    }

    // End frame and submit layers, even if layers is empty due to shouldRender = false
    XrFrameEndInfo frameEndInfo{ XR_TYPE_FRAME_END_INFO};
    frameEndInfo.displayTime = frameState.predictedDisplayTime;
    frameEndInfo.environmentBlendMode = XR_ENVIRONMENT_BLEND_MODE_OPAQUE;
    frameEndInfo.layerCount = (uint32_t)layers.size();
    frameEndInfo.layers = layers.data();
    CHK_XR(xrEndFrame(session, &frameEndInfo));
}

```

# Chapter 9. Session

XR\_DEFINE\_HANDLE(XrSession)

A session represents an application’s intention to display XR content to the user.

## 9.1. Session Lifecycle

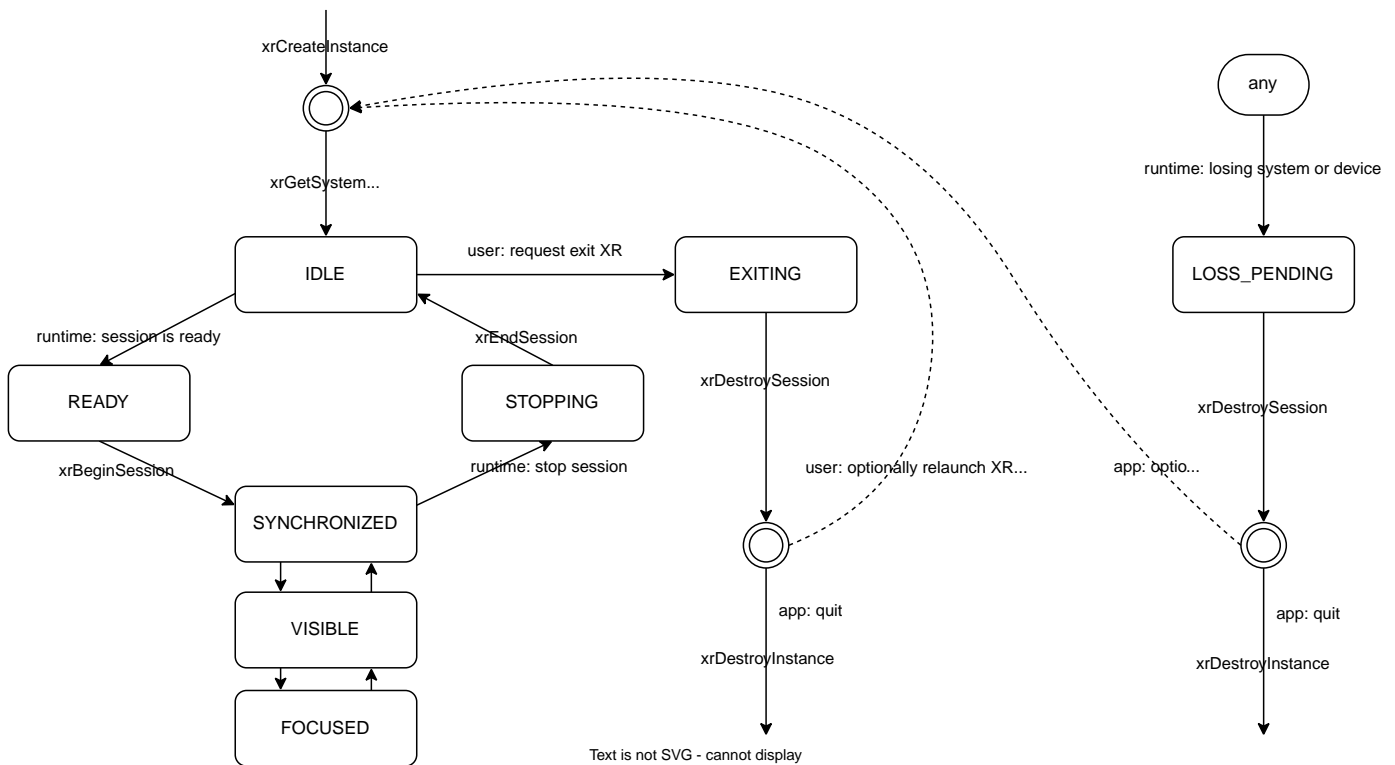


Figure 5. Session Life-cycle

A typical XR session coordinates the application and the runtime through session control functions and session state events.



1. The application creates a session by choosing a [system](#) and a graphics API and passing them into [xrCreateSession](#). The newly created session is in the [XR\\_SESSION\\_STATE\\_IDLE](#) state.
2. The application can regularly call [xrPollEvent](#) to monitor for session state changes via [XrEventDataSessionStateChanged](#) events.
3. When the runtime determines that the system is ready to start transitioning to this session’s XR content, the application receives a notification of session state change to [XR\\_SESSION\\_STATE\\_READY](#). Once the application is also ready to proceed and display its XR content, it calls [xrBeginSession](#) and [starts its frame loop](#), which

begins a [running session](#).

4. While the session is running, the application is expected to continuously execute its frame loop by calling [xrWaitFrame](#), [xrBeginFrame](#) and [xrEndFrame](#) each frame, establishing synchronization with the runtime. Once the runtime is synchronized with the application's frame loop and ready to display application's frames, the session moves into the `XR_SESSION_STATE_SYNCHRONIZED` state. In this state, the submitted frames will not be displayed or visible to the user yet.
5. When the runtime intends to display frames from the application, it notifies with `XR_SESSION_STATE_VISIBLE` state, and sets `XrFrameState::shouldRender` to `true` in [xrWaitFrame](#). The application should render XR content and submit the composition layers to [xrEndFrame](#).
6. When the runtime determines the application is eligible to receive XR inputs, e.g. motion controller or hand tracking inputs, it notifies with `XR_SESSION_STATE_FOCUSED` state. The application can expect to receive active [action inputs](#).
7. When the runtime determines the application has lost XR input focus, it moves the session state from `XR_SESSION_STATE_FOCUSED` to `XR_SESSION_STATE_VISIBLE` state. The application may need to change its own internal state while input is unavailable. Since the session is still visible, the application needs to render and submit frames at full frame rate, but may wish to change visually to indicate its input suspended state. When the runtime returns XR focus back to the application, it moves the session state back to `XR_SESSION_STATE_FOCUSED`.
8. When the runtime needs to end a [running session](#) due to the user closing or switching the application, the runtime will change the session state through appropriate intermediate ones and finally to `XR_SESSION_STATE_STOPPING`. When the application receives the `XR_SESSION_STATE_STOPPING` event, it should stop its frame loop and then call [xrEndSession](#) to tell the runtime to [stop the running session](#).
9. After [xrEndSession](#), the runtime transitions the session state to `XR_SESSION_STATE_IDLE`. If the XR session is temporarily paused in the background, the runtime will keep the session state at `XR_SESSION_STATE_IDLE` and later transition the session state back to `XR_SESSION_STATE_READY` when the XR session is resumed. If the runtime determines that its use of this XR session has concluded, it will transition the session state from `XR_SESSION_STATE_IDLE` to `XR_SESSION_STATE_EXITING`.
10. When the application receives the `XR_SESSION_STATE_EXITING` event, it releases the resources related to the session and calls [xrDestroySession](#).

A session is considered **running** after a successful call to [xrBeginSession](#) and remains running until any call is made to [xrEndSession](#). Certain functions are only valid to call when a session is running, such as [xrWaitFrame](#), or else the `XR_ERROR_SESSION_NOT_RUNNING` error **must** be returned by the runtime.

A session is considered **not running** before a successful call to [xrBeginSession](#) and becomes not

running again after any call is made to [xrEndSession](#). Certain functions are only valid to call when a session is not running, such as [xrBeginSession](#), or else the `XR_ERROR_SESSION_RUNNING` error **must** be returned by the runtime.

If an error is returned from [xrBeginSession](#), the session remains in its current running or not running state. Calling [xrEndSession](#) always transitions a session to the not running state, regardless of any errors returned.

Only running sessions may become focused sessions that receive XR input. When a session is **not running**, the application **must** not submit frames. This is important because without a running session, the runtime no longer has to spend resources on sub-systems (tracking etc.) that are no longer needed by the application.

An application **must** call [xrBeginSession](#) when the session is in the `XR_SESSION_STATE_READY` state, or `XR_ERROR_SESSION_NOT_READY` will be returned; it **must** call [xrEndSession](#) when the session is in the `XR_SESSION_STATE_STOPPING` state, otherwise `XR_ERROR_SESSION_NOT_STOPPING` will be returned. This is to allow the runtimes to seamlessly transition from one application's session to another.

The application **can** call [xrDestroySession](#) at any time during the session life cycle, however, it **must** stop using the [XrSession](#) handle immediately in all threads and stop using any related resources. Therefore, it's typically undesirable to destroy a **running session** and instead it's recommended to wait for `XR_SESSION_STATE_EXITING` to destroy a session.

## 9.2. Session Creation

To present graphical content on an output device, OpenXR applications need to pick a graphics API which is supported by the runtime. Unextended OpenXR does not support any graphics APIs natively but provides a number of extensions of which each runtime can support any subset. These extensions can be activated during [XrInstance](#) create time.

During [XrSession](#) creation the application **must** provide information about which graphics API it intends to use by adding an `XrGraphicsBinding*` struct of one (and only one) of the enabled graphics API extensions to the next chain of [XrSessionCreateInfo](#). The application **must** call the `xrGet*GraphicsRequirements` method (where `*` is a placeholder) provided by the chosen graphics API extension before attempting to create the session (for example, [xrGetD3D11GraphicsRequirementsKHR](#), [xrGetD3D12GraphicsRequirementsKHR](#), [xrGetOpenGLGraphicsRequirementsKHR](#), [xrGetVulkanGraphicsRequirementsKHR](#), [xrGetVulkanGraphicsRequirements2KHR](#) ).

Unless specified differently in the graphics API extension, the application is responsible for creating a valid graphics device binding based on the requirements returned by `xrGet*GraphicsRequirements` methods (for details refer to the extension specification of the graphics API).

The [xrCreateSession](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrCreateSession(
    XrInstance                instance,
    const XrSessionCreateInfo* createInfo,
    XrSession*                session);
```

## Parameter Descriptions

- `instance` is the instance from which `XrSessionCreateInfo::systemId` was retrieved.
- `createInfo` is a pointer to an `XrSessionCreateInfo` structure containing information about how to create the session.
- `session` is a pointer to a handle in which the created `XrSession` is returned.

Creates a session using the provided `createInfo` and returns a handle to that session. This session is created in the `XR_SESSION_STATE_IDLE` state, and a corresponding `XrEventDataSessionStateChanged` event to the `XR_SESSION_STATE_IDLE` state **must** be generated as the first such event for the new session.

The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` (`XR_ERROR_VALIDATION_FAILURE` may be returned due to legacy behavior) on calls to `xrCreateSession` if a function named like `xrGet*GraphicsRequirements` has not been called for the same `instance` and `XrSessionCreateInfo::systemId`. (See graphics binding extensions for details.)

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `createInfo` **must** be a pointer to a valid `XrSessionCreateInfo` structure
- `session` **must** be a pointer to an `XrSession` handle

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_SYSTEM_INVALID`
- `XR_ERROR_INITIALIZATION_FAILED`
- `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING`
- `XR_ERROR_GRAPHICS_DEVICE_INVALID`

The `XrSessionCreateInfo` structure is defined as:

```
typedef struct XrSessionCreateInfo {  
    XrStructureType      type;  
    const void*          next;  
    XrSessionCreateFlags createFlags;  
    XrSystemId           systemId;  
} XrSessionCreateInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR. Note that in most cases one graphics API extension specific struct needs to be in this next chain.
- `createFlags` identifies `XrSessionCreateFlags` that apply to the creation.
- `systemId` is the `XrSystemId` representing the system of devices to be used by this session.

## Valid Usage

- `systemId` **must** be a valid [XrSystemId](#) or `XR_ERROR_SYSTEM_INVALID` **must** be returned.
- `next`, unless otherwise specified via an extension, **must** contain exactly one graphics API binding structure (a structure whose name begins with “XrGraphicsBinding”) or `XR_ERROR_GRAPHICS_DEVICE_INVALID` **must** be returned.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SESSION_CREATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain. See also: [XrGraphicsBindingD3D11KHR](#), [XrGraphicsBindingD3D12KHR](#), [XrGraphicsBindingEGLMNDX](#), [XrGraphicsBindingOpenGLESAAndroidKHR](#), [XrGraphicsBindingOpenGLWaylandKHR](#), [XrGraphicsBindingOpenGLWin32KHR](#), [XrGraphicsBindingOpenGLXcbKHR](#), [XrGraphicsBindingOpenGLXlibKHR](#), [XrGraphicsBindingVulkanKHR](#), [XrHolographicWindowAttachmentMSFT](#), [XrSessionCreateInfoOverlayEXTX](#)
- `createFlags` **must** be `0`

The [XrSessionCreateInfo::createFlags](#) member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in [XrSessionCreateFlagBits](#).

```
typedef XrFlags64 XrSessionCreateFlags;
```

Valid bits for [XrSessionCreateFlags](#) are defined by [XrSessionCreateFlagBits](#).

```
// Flag bits for XrSessionCreateFlags
```

There are currently no session creation flags. This is reserved for future use.

The [xrDestroySession](#) function is defined as.

```
// Provided by XR_VERSION_1_0
XrResult xrDestroySession(
    XrSession session);
```

## Parameter Descriptions

- `session` is the session to destroy.

`XrSession` handles are destroyed using `xrDestroySession`. When an `XrSession` is destroyed, all handles that are children of that `XrSession` are also destroyed.

The application is responsible for ensuring that it has no calls using `session` in progress when the session is destroyed.

`xrDestroySession` can be called when the session is in any session state.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle

## Thread Safety

- Access to `session`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_HANDLE_INVALID`

## 9.3. Session Control

The `xrBeginSession` function is defined as:



```
// Provided by XR_VERSION_1_0
XrResult xrBeginSession(
    XrSession session,
    const XrSessionBeginInfo* beginInfo);
```

## Parameter Descriptions

- `session` is a valid `XrSession` handle.
- `beginInfo` is a pointer to an `XrSessionBeginInfo` structure.

When the application receives `XrEventDataSessionStateChanged` event with the `XR_SESSION_STATE_READY` state, the application **should** then call `xrBeginSession` to start rendering frames for display to the user.

After this function successfully returns, the session is considered to be running. The application **should** then start its frame loop consisting of some sequence of `xrWaitFrame`/`xrBeginFrame`/`xrEndFrame` calls.

If the session is already running when the application calls `xrBeginSession`, the runtime **must** return error `XR_ERROR_SESSION_RUNNING`. If the session is not running when the application calls `xrBeginSession`, but the session is not yet in the `XR_SESSION_STATE_READY` state, the runtime **must** return error `XR_ERROR_SESSION_NOT_READY`.

Note that a runtime **may** decide not to show the user any given frame from a session at any time, for example if the user has switched to a different application's running session. The application should check whether `xrWaitFrame` returns `XrFrameState::shouldRender` set to true before rendering a given frame to determine whether that frame will be visible to the user.

Runtime session frame state **must** start in a reset state when a session transitions to `running` so that no state is carried over from when the same session was previously running. Frame state in this context includes `xrWaitFrame`, `xrBeginFrame`, and `xrEndFrame` call order enforcement.

If `XrSessionBeginInfo::primaryViewConfigurationType` in `beginInfo` is not supported by the `XrSystemId` used to create the `session`, the runtime **must** return `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `beginInfo` **must** be a pointer to a valid `XrSessionBeginInfo` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`
- `XR_ERROR_SESSION_RUNNING`
- `XR_ERROR_SESSION_NOT_READY`

The `XrSessionBeginInfo` structure is defined as:

```
typedef struct XrSessionBeginInfo {  
    XrStructureType      type;  
    const void*         next;  
    XrViewConfigurationType primaryViewConfigurationType;  
} XrSessionBeginInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `primaryViewConfigurationType` is the `XrViewConfigurationType` to use during this session to provide images for the form factor's primary displays.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_SESSION_BEGIN_INFO`
- **next** **must** be `NULL` or a valid pointer to the [next](#) structure in a structure chain. See also: [XrSecondaryViewConfigurationSessionBeginInfoMSFT](#)
- **primaryViewConfigurationType** **must** be a valid [XrViewConfigurationType](#) value

The `xrEndSession` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEndSession(
    XrSession session);
```

## Parameter Descriptions

- **session** is a handle to a [running XrSession](#).

When the application receives [XrEventDataSessionStateChanged](#) event with the `XR_SESSION_STATE_STOPPING` state, the application should stop its frame loop and then call `xrEndSession` to end the [running](#) session. This function signals to the runtime that the application will no longer call `xrWaitFrame`, `xrBeginFrame` or `xrEndFrame` from any thread allowing the runtime to safely transition the session to `XR_SESSION_STATE_IDLE`. The application **must** also avoid reading input state or sending haptic output after calling `xrEndSession`.

If the session **is not running** when the application calls `xrEndSession`, the runtime **must** return error `XR_ERROR_SESSION_NOT_RUNNING`. If the session **is still running** when the application calls `xrEndSession`, but the session is not yet in the `XR_SESSION_STATE_STOPPING` state, the runtime **must** return error `XR_ERROR_SESSION_NOT_STOPPING`.

If the application wishes to exit a running session, the application can call `xrRequestExitSession` so that the session transitions from `XR_SESSION_STATE_IDLE` to `XR_SESSION_STATE_EXITING`.

## Valid Usage (Implicit)

- **session** **must** be a valid [XrSession](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SESSION_NOT_STOPPING`
- `XR_ERROR_SESSION_NOT_RUNNING`

When an application wishes to exit a **running** session, it **can** call `xrRequestExitSession`, requesting that the runtime transition through the various intermediate session states including `XR_SESSION_STATE_STOPPING` to `XR_SESSION_STATE_EXITING`.

On platforms where an application's lifecycle is managed by the system, session state changes may be implicitly triggered by application lifecycle state changes. On such platforms, using platform-specific methods to alter application lifecycle state may be the preferred method of provoking session state changes. The behavior of `xrRequestExitSession` is not altered, however explicit session exit **may** not interact with the platform-specific application lifecycle.

The `xrRequestExitSession` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrRequestExitSession(
    XrSession session);
```

## Parameter Descriptions

- `session` is a handle to a running `XrSession`.

If `session` is **not running** when `xrRequestExitSession` is called, `XR_ERROR_SESSION_NOT_RUNNING` **must** be returned.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SESSION_NOT_RUNNING`

## 9.4. Session States

While events can be expanded upon, there are a minimum set of lifecycle events which can occur which all OpenXR applications must be aware of. These events are detailed below.

### 9.4.1. XrEventDataSessionStateChanged

The [XrEventDataSessionStateChanged](#) structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrEventDataSessionStateChanged {
    XrStructureType    type;
    const void*        next;
    XrSession           session;
    XrSessionState     state;
    XrTime              time;
} XrEventDataSessionStateChanged;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `session` is the [XrSession](#) which has changed state.
- `state` is the current [XrSessionState](#) of the `session`.
- `time` is an [XrTime](#) which indicates the time of the state change.

Receiving the [XrEventDataSessionStateChanged](#) event structure indicates that the application has changed lifecycle state.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_EVENT_DATA_SESSION_STATE_CHANGED`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrSessionState](#) enumerates the possible session lifecycle states:

```
typedef enum XrSessionState {
    XR_SESSION_STATE_UNKNOWN = 0,
    XR_SESSION_STATE_IDLE = 1,
    XR_SESSION_STATE_READY = 2,
    XR_SESSION_STATE_SYNCHRONIZED = 3,
    XR_SESSION_STATE_VISIBLE = 4,
    XR_SESSION_STATE_FOCUSED = 5,
    XR_SESSION_STATE_STOPPING = 6,
    XR_SESSION_STATE_LOSS_PENDING = 7,
    XR_SESSION_STATE_EXITING = 8,
    XR_SESSION_STATE_MAX_ENUM = 0x7FFFFFFF
} XrSessionState;
```

## Enumerant Descriptions

- `XR_SESSION_STATE_UNKNOWN`. An unknown state. The runtime **must** not return this value in an `XrEventDataSessionStateChanged` event.
- `XR_SESSION_STATE_IDLE`. The initial state after calling `xrCreateSession` or returned to after calling `xrEndSession`.
- `XR_SESSION_STATE_READY`. The application is ready to call `xrBeginSession` and `sync its frame loop with the runtime`.
- `XR_SESSION_STATE_SYNCHRONIZED`. The application has synced its frame loop with the runtime but is not visible to the user.
- `XR_SESSION_STATE_VISIBLE`. The application has `synced its frame loop with the runtime` and is visible to the user but cannot receive XR input.
- `XR_SESSION_STATE_FOCUSED`. The application has `synced its frame loop with the runtime`, is visible to the user and can receive XR input.
- `XR_SESSION_STATE_STOPPING`. The application should exit its frame loop and call `xrEndSession`.
- `XR_SESSION_STATE_LOSS_PENDING`. The session is in the process of being lost. The application should destroy the current session and can optionally recreate it.
- `XR_SESSION_STATE_EXITING`. The application should end its XR experience and not automatically restart it.

The `XR_SESSION_STATE_UNKNOWN` state **must** not be returned by the runtime, and is only defined to avoid `0` being a valid state.

Receiving the `XR_SESSION_STATE_IDLE` state indicates that the runtime considers the session is idle. Applications in this state **should** minimize resource consumption but continue to call `xrPollEvent` at some reasonable cadence.

Receiving the `XR_SESSION_STATE_READY` state indicates that the runtime desires the application to prepare rendering resources, begin its session and synchronize its frame loop with the runtime.

The application does this by successfully calling `xrBeginSession` and then running its frame loop by calling `xrWaitFrame`, `xrBeginFrame` and `xrEndFrame` in a loop. If the runtime wishes to return the session to the `XR_SESSION_STATE_IDLE` state, it **must** wait until the application calls `xrBeginSession`. After returning from the `xrBeginSession` call, the runtime may then immediately transition forward through the `XR_SESSION_STATE_SYNCHRONIZED` state to the `XR_SESSION_STATE_STOPPING` state, to request that the application end this session. If the system supports a user engagement sensor and runtime is in `XR_SESSION_STATE_IDLE` state, the runtime **may** wait until the user starts engaging with the device before transitioning to the `XR_SESSION_STATE_READY` state.

Receiving the `XR_SESSION_STATE_SYNCHRONIZED` state indicates that the application has `synchronized its frame loop with the runtime`, but its frames are not visible to the user. The application **should** continue

running its frame loop by calling `xrWaitFrame`, `xrBeginFrame` and `xrEndFrame`, although it should avoid heavy GPU work so that other visible applications can take CPU and GPU precedence. The application can save resources here by skipping rendering and not submitting any composition layers until `xrWaitFrame` returns an `XrFrameState` with `shouldRender` set to true. A runtime **may** use this frame synchronization to facilitate seamless switching from a previous XR application to this application on a frame boundary.

Receiving the `XR_SESSION_STATE_VISIBLE` state indicates that the application has [synchronized its frame loop with the runtime](#), and the session's frames will be visible to the user, but the session is not eligible to receive XR input. An application may be visible but not have focus, for example when the runtime is composing a modal pop-up on top of the application's rendered frames. The application **should** continue running its frame loop, rendering and submitting its composition layers, although it may wish to pause its experience, as users cannot interact with the application at this time. It is important for applications to continue rendering when visible, even when they do not have focus, so the user continues to see something reasonable underneath modal pop-ups. Runtimes **should** make input actions inactive while the application is unfocused, and applications should react to an inactive input action by skipping rendering of that action's input avatar (depictions of hands or other tracked objects controlled by the user).

Receiving the `XR_SESSION_STATE_FOCUSED` state indicates that the application has [synchronized its frame loop with the runtime](#), the session's frames will be visible to the user, and the session is eligible to receive XR input. The runtime **should** only give one session XR input focus at any given time. The application **should** be running its frame loop, rendering and submitting composition layers, including input avatars (depictions of hands or other tracked objects controlled by the user) for any input actions that are active. The runtime **should** avoid rendering its own input avatars when an application is focused, unless input from a given source is being captured by the runtime at the moment.

Receiving the `XR_SESSION_STATE_STOPPING` state indicates that the runtime has determined that the application should halt its rendering loop. Applications **should** exit their rendering loop and call `xrEndSession` when in this state. A possible reason for this would be to minimize contention between multiple applications. If the system supports a user engagement sensor and the session is running, the runtime **may** transition to the `XR_SESSION_STATE_STOPPING` state when the user stops engaging with the device.

Receiving the `XR_SESSION_STATE_EXITING` state indicates the runtime wishes the application to terminate its XR experience, typically due to a user request via a runtime user interface. Applications **should** gracefully end their process when in this state if they do not have a non-XR user experience.

Receiving the `XR_SESSION_STATE_LOSS_PENDING` state indicates the runtime is no longer able to operate with the current session, for example due to the loss of a display hardware connection. An application **should** call `xrDestroySession` and **may** end its process or decide to poll `xrGetSystem` at some reasonable cadence to get a new `XrSystemId`, and re-initialize all graphics resources related to the new system, and then create a new session using `xrCreateSession`. After the event is queued, subsequent calls to functions that accept `XrSession` parameters **must** no longer return any success code other than `XR_SESSION_LOSS_PENDING` for the given `XrSession` handle. The `XR_SESSION_LOSS_PENDING` success result is returned for an unspecified grace period of time, and the functions that return it simulate success in



their behavior. If the runtime has no reasonable way to successfully complete a given function (e.g. `xrCreateSwapchain`) when a lost session is pending, or if the runtime is not able to provide the application a grace period, the runtime **may** return `XR_ERROR_SESSION_LOST`. Thereafter, functions which accept `XrSession` parameters for the lost session **may** return `XR_ERROR_SESSION_LOST` to indicate that the function failed and the given session was lost. The `XrSession` handle and child handles are henceforth unusable and **should** be destroyed by the application in order to immediately free up resources associated with those handles.

# Chapter 10. Rendering

## 10.1. Swapchain Image Management

```
XR_DEFINE_HANDLE(XrSwapchain)
```

Normal XR applications will want to present rendered images to the user. To allow this, the runtime provides images organized in swapchains for the application to render into. The runtime **must** allow applications to create multiple swapchains.

Swapchain image format support by the runtime is specified by the `xrEnumerateSwapchainFormats` function. Runtimes **should** support `R8G8B8A8` and `R8G8B8A8 sRGB` formats if possible.

Swapchain images **can** be 2D or 2D Array.

Rendering operations involving composition of submitted layers are assumed to be internally performed by the runtime in linear color space. Images submitted in sRGB color space **must** be created using an API-specific sRGB format (e.g. `DXGI_FORMAT_R8G8B8A8_UNORM_SRGB`, `GL_SRGB8_ALPHA8`, `VK_FORMAT_R8G8B8A8_SRGB`) to apply automatic sRGB-to-linear conversion when read by the runtime. All other formats will be treated as linear values.

### Note



OpenXR applications **should** avoid submitting linear encoded 8 bit color data (e.g. `DXGI_FORMAT_R8G8B8A8_UNORM`) whenever possible as it **may** result in color banding.

Gritz, L. and d'Eon, E. 2007. The Importance of Being Linear. In: H. Nguyen, ed., *GPU Gems 3*. Addison-Wesley Professional. <https://developer.nvidia.com/gpugems/gpugems3/part-iv-image-effects/chapter-24-importance-being-linear>

### Note



DXGI resources will be created with their associated TYPELESS format, but the runtime will use the application-specified format for reading the data.

The `xrEnumerateSwapchainFormats` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateSwapchainFormats(
    XrSession          session,
    uint32_t          formatCapacityInput,
    uint32_t*         formatCountOutput,
    int64_t*          formats);
```

## Parameter Descriptions

- `session` is the session that enumerates the supported formats.
- `formatCapacityInput` is the capacity of the `formats`, or 0 to retrieve the required capacity.
- `formatCountOutput` is a pointer to the count of `uint64_t` formats written, or a pointer to the required capacity in the case that `formatCapacityInput` is insufficient.
- `formats` is a pointer to an array of `int64_t` format ids, but **can** be `NULL` if `formatCapacityInput` is 0. The format ids are specific to the specified graphics API.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `formats` size.

`xrEnumerateSwapchainFormats` enumerates the texture formats supported by the current session. The type of formats returned are dependent on the graphics API specified in `xrCreateSession`. For example, if a DirectX graphics API was specified, then the enumerated formats correspond to the DXGI formats, such as `DXGI_FORMAT_R8G8B8A8_UNORM_SRGB`. Texture formats **should** be in order from highest to lowest runtime preference. The application **should** use the highest preference format that it supports for optimal performance and quality.

With an OpenGL-based graphics API, the texture formats correspond to OpenGL internal formats.

With a Direct3D-based graphics API, `xrEnumerateSwapchainFormats` never returns typeless formats (e.g. `DXGI_FORMAT_R8G8B8A8_TYPELESS`). Only concrete formats are returned, and only concrete formats **may** be specified by applications for swapchain creation.

Runtimes **must** always return identical buffer contents from this enumeration for the lifetime of the session.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `formatCountOutput` **must** be a pointer to a `uint32_t` value
- If `formatCapacityInput` is not 0, `formats` **must** be a pointer to an array of `formatCapacityInput` `int64_t` values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `xrCreateSwapchain` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrCreateSwapchain(
    XrSession session,
    const XrSwapchainCreateInfo* createInfo,
    XrSwapchain* swapchain);
```

## Parameter Descriptions

- `session` is the session that creates the image.
- `createInfo` is a pointer to an `XrSwapchainCreateInfo` structure containing parameters to be used to create the image.
- `swapchain` is a pointer to a handle in which the created `XrSwapchain` is returned.

Creates an `XrSwapchain` handle. The returned swapchain handle **may** be subsequently used in API calls. Multiple `XrSwapchain` handles **may** exist simultaneously, up to some limit imposed by the runtime. The `XrSwapchain` handle **must** be eventually freed via the `xrDestroySwapchain` function. The runtime **must** return `XR_ERROR_SWAPCHAIN_FORMAT_UNSUPPORTED` if the image format specified in the `XrSwapchainCreateInfo` is unsupported. The runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if any bit of the create or usage flags specified in the `XrSwapchainCreateInfo` is unsupported.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrSwapchainCreateInfo` structure
- `swapchain` **must** be a pointer to an `XrSwapchain` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_SWAPCHAIN_FORMAT_UNSUPPORTED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrSwapchainCreateInfo` structure is defined as:

```
typedef struct XrSwapchainCreateInfo {
    XrStructureType      type;
    const void*          next;
    XrSwapchainCreateFlags createFlags;
    XrSwapchainUsageFlags usageFlags;
    int64_t              format;
    uint32_t             sampleCount;
    uint32_t             width;
    uint32_t             height;
    uint32_t             faceCount;
    uint32_t             arraySize;
    uint32_t             mipCount;
} XrSwapchainCreateInfo;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- **createFlags** is a bitmask of [XrSwapchainCreateFlagBits](#) describing additional properties of the swapchain.
- **usageFlags** is a bitmask of [XrSwapchainUsageFlagBits](#) describing the intended usage of the swapchain's images. The usage flags define how the corresponding graphics API objects are created. A mismatch **may** result in swapchain images that do not support the application's usage.
- **format** is a graphics API-specific texture format identifier. For example, if the graphics API specified in [xrCreateSession](#) is Vulkan, then this format is a Vulkan format such as `VK_FORMAT_R8G8B8A8_SRGB`. The format identifies the format that the runtime will interpret the texture as upon submission. Valid formats are indicated by [xrEnumerateSwapchainFormats](#).
- **sampleCount** is the number of sub-data element samples in the image, **must** not be `0` or greater than the graphics API's maximum limit.
- **width** is the width of the image, **must** not be `0` or greater than the graphics API's maximum limit.
- **height** is the height of the image, **must** not be `0` or greater than the graphics API's maximum limit.
- **faceCount** is the number of faces, which **must** be either `6` (for cubemaps) or `1`.
- **arraySize** is the number of array layers in the image or `1` for a 2D image, **must** not be `0` or greater than the graphics API's maximum limit.
- **mipCount** describes the number of levels of detail available for minified sampling of the image, **must** not be `0` or greater than the graphics API's maximum limit.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_SWAPCHAIN_CREATE_INFO`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrAndroidSurfaceSwapchainCreateInfoFB](#), [XrSecondaryViewConfigurationSwapchainCreateInfoMSFT](#), [XrSwapchainCreateInfoFoveationFB](#), [XrVulkanSwapchainCreateInfoMETA](#)
- **createFlags** **must** be `0` or a valid combination of [XrSwapchainCreateFlagBits](#) values
- **usageFlags** **must** be `0` or a valid combination of [XrSwapchainUsageFlagBits](#) values

The `XrSwapchainCreateInfo::createFlags` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in `XrSwapchainCreateFlagBits`.

```
typedef XrFlags64 XrSwapchainCreateFlags;
```

Valid bits for `XrSwapchainCreateFlags` are defined by `XrSwapchainCreateFlagBits`, which is specified as:

```
// Flag bits for XrSwapchainCreateFlags
static const XrSwapchainCreateFlags XR_SWAPCHAIN_CREATE_PROTECTED_CONTENT_BIT =
0x00000001;
static const XrSwapchainCreateFlags XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT = 0x00000002;
```

The flag bits have the following meanings:

### Flag Descriptions

- `XR_SWAPCHAIN_CREATE_PROTECTED_CONTENT_BIT` indicates that the swapchain's images will be protected from CPU access, using a mechanism such as Vulkan protected memory.
- `XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT` indicates that the application will acquire and release only one image to this swapchain over its entire lifetime. The runtime **must** allocate only one swapchain image.

A runtime **may** implement any of these, but is not required to. A runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateSwapchain` if an `XrSwapchainCreateFlags` bit is requested but not implemented.

`XrSwapchainUsageFlags` specify the intended usage of the swapchain images. The `XrSwapchainCreateInfo::usageFlags` member is of this type, and contains a bitwise-OR of one or more of the bits defined in `XrSwapchainUsageFlagBits`.

```
typedef XrFlags64 XrSwapchainUsageFlags;
```

When images are created, the runtime needs to know how the images are used in a way that requires more information than simply the image format. The `XrSwapchainCreateInfo` passed to `xrCreateSwapchain` **must** match the intended usage.



Flags include:

```
// Flag bits for XrSwapchainUsageFlags
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_COLOR_ATTACHMENT_BIT = 0x00000001;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT =
0x00000002;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_UNORDERED_ACCESS_BIT = 0x00000004;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT = 0x00000008;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_TRANSFER_DST_BIT = 0x00000010;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_SAMPLED_BIT = 0x00000020;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT = 0x00000040;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_MND =
0x00000080;
static const XrSwapchainUsageFlags XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR =
0x00000080; // alias of XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_MND
```

The flag bits have the following meanings:

### Flag Descriptions

- `XR_SWAPCHAIN_USAGE_COLOR_ATTACHMENT_BIT` — Specifies that the image **may** be a color rendering target.
- `XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` — Specifies that the image **may** be a depth/stencil rendering target.
- `XR_SWAPCHAIN_USAGE_UNORDERED_ACCESS_BIT` — Specifies that the image **may** be accessed out of order and that access **may** be via atomic operations.
- `XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT` — Specifies that the image **may** be used as the source of a transfer operation.
- `XR_SWAPCHAIN_USAGE_TRANSFER_DST_BIT` — Specifies that the image **may** be used as the destination of a transfer operation.
- `XR_SWAPCHAIN_USAGE_SAMPLED_BIT` — Specifies that the image **may** be sampled by a shader.
- `XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT` — Specifies that the image **may** be reinterpreted as another image format.
- `XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_MND` — Specifies that the image **may** be used as a input attachment. (Added by the `XR_MND_swapchain_usage_input_attachment_bit` extension)
- `XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR` — Specifies that the image **may** be used as a input attachment. (Added by the `XR_KHR_swapchain_usage_input_attachment_bit` extension)

The number of images in each swapchain is implementation-defined except in the case of a static

swapchain. To obtain the number of images actually allocated, call [xrEnumerateSwapchainImages](#).

With a Direct3D-based graphics API, the swapchain returned by [xrCreateSwapchain](#) will be a typeless format if the requested format has a typeless analogue. Applications are required to reinterpret the swapchain as a compatible non-typeless type. Upon submitting such swapchains to the runtime, they are interpreted as the format specified by the application in the [XrSwapchainCreateInfo](#).

Swapchains will be created with graphics API-specific flags appropriate to the type of underlying image and its usage.

Runtimes **must** honor underlying graphics API limits when creating resources.

[xrEnumerateSwapchainFormats](#) never returns typeless formats (e.g. `DXGI_FORMAT_R8G8B8A8_TYPELESS`). Only concrete formats are returned, and only concrete formats **may** be specified by applications for swapchain creation.

The [xrDestroySwapchain](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrDestroySwapchain(
    XrSwapchain          swapchain);
```

### Parameter Descriptions

- `swapchain` is the swapchain to destroy.

All submitted graphics API commands that refer to `swapchain` **must** have completed execution. Runtimes **may** continue to utilize swapchain images after [xrDestroySwapchain](#) is called.

### Valid Usage (Implicit)

- `swapchain` **must** be a valid [XrSwapchain](#) handle

### Thread Safety

- Access to `swapchain`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_HANDLE_INVALID`

Swapchain images are acquired, waited on, and released by index, but the number of images in a swapchain is implementation-defined. Additionally, rendering to images requires access to the underlying image primitive of the graphics API being used. Applications **may** query and cache the images at any time after swapchain creation.

The `xrEnumerateSwapchainImages` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateSwapchainImages(
    XrSwapchain                swapchain,
    uint32_t                   imageCapacityInput,
    uint32_t*                   imageCountOutput,
    XrSwapchainImageBaseHeader* images);
```

## Parameter Descriptions

- `swapchain` is the `XrSwapchain` to get images from.
- `imageCapacityInput` is the capacity of the `images` array, or 0 to indicate a request to retrieve the required capacity.
- `imageCountOutput` is a pointer to the count of `images` written, or a pointer to the required capacity in the case that `imageCapacityInput` is insufficient.
- `images` is a pointer to an array of graphics API-specific `XrSwapchainImage` structures, all of the same type, based on `XrSwapchainImageBaseHeader`. It **can** be `NULL` if `imageCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `images` size.

Fills an array of graphics API-specific `XrSwapchainImage` structures. The resources **must** be constant and valid for the lifetime of the `XrSwapchain`.

Runtimes **must** always return identical buffer contents from this enumeration for the lifetime of the swapchain.

Note: `images` is a pointer to an array of structures of graphics API-specific type, not an array of structure pointers.

The pointer submitted as `images` will be treated as an array of the expected graphics API-specific type based on the graphics API used at session creation time. If the `type` member of any array element accessed in this way does not match the expected value, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.



*Note*

Under a typical memory model, a runtime **must** treat the supplied pointer as an opaque blob beginning with `XrSwapchainImageBaseHeader`, until after it has verified the `XrSwapchainImageBaseHeader::type`.

### Valid Usage (Implicit)

- `swapchain` **must** be a valid `XrSwapchain` handle
- `imageCountOutput` **must** be a pointer to a `uint32_t` value
- If `imageCapacityInput` is not `0`, `images` **must** be a pointer to an array of `imageCapacityInput` `XrSwapchainImageBaseHeader`-based structures. See also: `XrSwapchainImageD3D11KHR`, `XrSwapchainImageD3D12KHR`, `XrSwapchainImageOpenGLESKHR`, `XrSwapchainImageOpenGLKHR`, `XrSwapchainImageVulkanKHR`

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `XrSwapchainImageBaseHeader` structure is defined as:

```
typedef struct XrSwapchainImageBaseHeader {
    XrStructureType    type;
    void*              next;
} XrSwapchainImageBaseHeader;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure. This base structure itself has no associated `XrStructureType` value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

The `XrSwapchainImageBaseHeader` is a base structure that is extended by graphics API-specific `XrSwapchainImage*` child structures.

## Valid Usage (Implicit)

- `type` **must** be one of the following `XrStructureType` values:  
`XR_TYPE_SWAPCHAIN_IMAGE_D3D11_KHR`, `XR_TYPE_SWAPCHAIN_IMAGE_D3D12_KHR`,  
`XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_ES_KHR`, `XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_KHR`,  
`XR_TYPE_SWAPCHAIN_IMAGE_VULKAN_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

Before an application builds graphics API command buffers that refer to an image in a swapchain, it **must** acquire the image from the swapchain. The acquire operation determines the index of the next image to be used in the swapchain. The order in which images are acquired is undefined. The runtime **must** allow the application to acquire more than one image from a single (non-static) swapchain at a time, for example if the application implements a multiple frame deep rendering pipeline.

The `xrAcquireSwapchainImage` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrAcquireSwapchainImage(
    XrSwapchain                swapchain,
    const XrSwapchainImageAcquireInfo* acquireInfo,
    uint32_t*                  index);
```

## Parameter Descriptions

- `swapchain` is the swapchain from which to acquire an image.
- `acquireInfo` exists for extensibility purposes, it is `NULL` or a pointer to a valid `XrSwapchainImageAcquireInfo`.
- `index` is the returned image index that has been acquired.

Acquires the image corresponding to the `index` position in the array returned by `xrEnumerateSwapchainImages`. The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if the next available index has already been acquired and not yet released with `xrReleaseSwapchainImage`. If the `swapchain` was created with the `XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT` set in `XrSwapchainCreateInfo::createFlags`, this function **must** not have been previously called for this swapchain. The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if a `swapchain` created with the `XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT` set in `XrSwapchainCreateInfo::createFlags` and this function has been successfully called previously for this swapchain.

This function only provides the index of the swapchain image, for example for use in recording command buffers. It does not wait for the image to be usable by the application. The application **must** call `xrWaitSwapchainImage` for each "acquire" call before submitting graphics commands that write to the image.

## Valid Usage (Implicit)

- `swapchain` **must** be a valid `XrSwapchain` handle
- If `acquireInfo` is not `NULL`, `acquireInfo` **must** be a pointer to a valid `XrSwapchainImageAcquireInfo` structure
- `index` **must** be a pointer to a `uint32_t` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_CALL_ORDER_INVALID`

The `XrSwapchainImageAcquireInfo` structure is defined as:

```
typedef struct XrSwapchainImageAcquireInfo {  
    XrStructureType    type;  
    const void*        next;  
} XrSwapchainImageAcquireInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

Because this structure only exists to support extension-specific structures, `xrAcquireSwapchainImage` will accept a `NULL` argument for `xrAcquireSwapchainImage::acquireInfo` for applications that are not using any relevant extensions.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_ACQUIRE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `xrWaitSwapchainImage` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrWaitSwapchainImage(
    XrSwapchain          swapchain,
    const XrSwapchainImageWaitInfo* waitInfo);
```

### Parameter Descriptions

- `swapchain` is the swapchain from which to wait for an image.
- `waitInfo` is a pointer to an `XrSwapchainImageWaitInfo` structure.

Before an application begins writing to a swapchain image, it **must** first wait on the image, to avoid writing to it before the compositor has finished reading from it. `xrWaitSwapchainImage` will implicitly wait on the oldest acquired swapchain image which has not yet been successfully waited on. Once a swapchain image has been successfully waited on without timeout, the app **must** release before waiting on the next acquired swapchain image.

This function **may** block for longer than the timeout specified in `XrSwapchainImageWaitInfo` due to scheduling or contention.

If the timeout expires without the image becoming available for writing, `XR_TIMEOUT_EXPIRED` **must** be returned. If `xrWaitSwapchainImage` returns `XR_TIMEOUT_EXPIRED`, the next call to `xrWaitSwapchainImage` will wait on the same image index again until the function succeeds with `XR_SUCCESS`. Note that this is not an error code; `XR_SUCCEEDED(XR_TIMEOUT_EXPIRED)` is `true`.

The runtime **must** eventually relinquish ownership of a swapchain image to the application and **must** not block indefinitely.

The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if no image has been acquired by calling `xrAcquireSwapchainImage`.

### Valid Usage (Implicit)

- `swapchain` **must** be a valid `XrSwapchain` handle
- `waitInfo` **must** be a pointer to a valid `XrSwapchainImageWaitInfo` structure



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_TIMEOUT_EXPIRED`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_CALL_ORDER_INVALID`

The [XrSwapchainImageWaitInfo](#) structure describes a swapchain image wait operation. It is defined as:

```
typedef struct XrSwapchainImageWaitInfo {  
    XrStructureType    type;  
    const void*        next;  
    XrDuration          timeout;  
} XrSwapchainImageWaitInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `timeout` indicates how many nanoseconds the call **may** block waiting for the image to become available for writing.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_WAIT_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

Once an application is done submitting commands that reference the swapchain image, the application **must** release the swapchain image. `xrReleaseSwapchainImage` will implicitly release the oldest swapchain image which has been acquired. The swapchain image **must** have been successfully waited on without timeout before it is released. `xrEndFrame` will use the most recently released swapchain image. In each frame submitted to the compositor, only one image index from each swapchain will be used. Note that in case the swapchain contains 2D image arrays, one array is referenced per swapchain index and thus the whole image array **may** be used in one frame.

The `xrReleaseSwapchainImage` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrReleaseSwapchainImage(
    XrSwapchain                swapchain,
    const XrSwapchainImageReleaseInfo* releaseInfo);
```

## Parameter Descriptions

- `swapchain` is the [XrSwapchain](#) from which to release an image.
- `releaseInfo` exists for extensibility purposes, it is `NULL` or a pointer to a valid [XrSwapchainImageReleaseInfo](#).

If the `swapchain` was created with the `XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT` set in `XrSwapchainCreateInfo::createFlags` structure, this function **must** not have been previously called for this swapchain.

The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if no image has been waited on by calling `xrWaitSwapchainImage`.

## Valid Usage (Implicit)

- `swapchain` **must** be a valid [XrSwapchain](#) handle
- If `releaseInfo` is not `NULL`, `releaseInfo` **must** be a pointer to a valid [XrSwapchainImageReleaseInfo](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_CALL_ORDER_INVALID`

The `XrSwapchainImageReleaseInfo` structure is defined as:

```
typedef struct XrSwapchainImageReleaseInfo {  
    XrStructureType    type;  
    const void*        next;  
} XrSwapchainImageReleaseInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

Because this structure only exists to support extension-specific structures, `xrReleaseSwapchainImage` will accept a `NULL` argument for `xrReleaseSwapchainImage::releaseInfo` for applications that are not using any relevant extensions.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_RELEASE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 10.2. View and Projection State

An application uses `xrLocateViews` to retrieve the viewer pose and projection parameters needed to render each view for use in a composition projection layer.

The `xrLocateViews` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrLocateViews(
    XrSession                session,
    const XrViewLocateInfo*  viewLocateInfo,
    XrViewState*             viewState,
    uint32_t                 viewCapacityInput,
    uint32_t*                viewCountOutput,
    XrView*                  views);
```

### Parameter Descriptions

- `session` is a handle to the provided `XrSession`.
- `viewLocateInfo` is a pointer to a valid `XrViewLocateInfo` structure.
- `viewState` is the output structure with the viewer state information.
- `viewCapacityInput` is an input parameter which specifies the capacity of the `views` array. The required capacity **must** be same as defined by the corresponding `XrViewConfigurationType`.
- `viewCountOutput` is an output parameter which identifies the valid count of `views`.
- `views` is an array of `XrView`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `views` size.

The `xrLocateViews` function returns the view and projection info for a particular display time. This time is typically the target 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.

`xrLocateViews` returns an array of `XrView` elements, one for each view of the specified view configuration type, along with an `XrViewState` containing additional state data shared across all views. The eye each view corresponds to is statically defined in `XrViewConfigurationType` in case the application wants to apply eye-specific rendering traits. The `XrViewState` and `XrView` member data

may change on subsequent calls to `xrLocateViews`, and so applications **must** not assume it to be constant.

If an application gives a `viewLocateInfo` with a `XrViewLocateInfo::viewConfigurationType` that was not passed in the session's call to `xrBeginSession` via the `XrSessionBeginInfo::primaryViewConfigurationType`, or enabled through an extension, then the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

### Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `viewLocateInfo` **must** be a pointer to a valid `XrViewLocateInfo` structure
- `viewState` **must** be a pointer to an `XrViewState` structure
- `viewCountOutput` **must** be a pointer to a `uint32_t` value
- If `viewCapacityInput` is not 0, `views` **must** be a pointer to an array of `viewCapacityInput` `XrView` structures

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`
- `XR_ERROR_TIME_INVALID`

The `XrViewLocateInfo` structure is defined as:

```
typedef struct XrViewLocateInfo {
    XrStructureType      type;
    const void*          next;
    XrViewConfigurationType viewConfigurationType;
    XrTime                displayTime;
    XrSpace                space;
} XrViewLocateInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `viewConfigurationType` is [XrViewConfigurationType](#) to query for.
- `displayTime` is the time for which the view poses are predicted.
- `space` is the [XrSpace](#) in which the `pose` in each [XrView](#) is expressed.

The [XrViewLocateInfo](#) structure contains the display time and space used to locate the view [XrView](#) structures.

The runtime **must** return error `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED` if the given `viewConfigurationType` is not one of the supported type reported by [xrEnumerateViewConfigurations](#).

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_VIEW_LOCATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrViewLocateFoveatedRenderingVARJO](#)
- `viewConfigurationType` **must** be a valid [XrViewConfigurationType](#) value
- `space` **must** be a valid [XrSpace](#) handle

The [XrView](#) structure is defined as:

```
typedef struct XrView {
    XrStructureType    type;
    void*              next;
    XrPosef            pose;
    XrFovf             fov;
} XrView;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `pose` is an [XrPosef](#) defining 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.

The [XrView](#) structure contains view pose and projection state necessary to render a single projection view in the view configuration.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_VIEW`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrViewState](#) structure is defined as:

```
typedef struct XrViewState {
    XrStructureType    type;
    void*              next;
    XrViewStateFlags   viewStateFlags;
} XrViewState;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `viewStateFlags` is a bitmask of [XrViewStateFlagBits](#) indicating state for all views.

The [XrViewState](#) contains additional view state from [xrLocateViews](#) common to all views of the active view configuration.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_VIEW_STATE`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `viewStateFlags` **must** be `0` or a valid combination of [XrViewStateFlagBits](#) values

The [XrViewStateFlags](#) specifies the validity and quality of the corresponding [XrView](#) array returned by [xrLocateViews](#). The [XrViewState::viewStateFlags](#) member is of this type, and contains a bitwise-OR of zero or more of the bits defined in [XrViewStateFlagBits](#).

```
typedef XrFlags64 XrViewStateFlags;
```

Valid bits for [XrViewStateFlags](#) are defined by [XrViewStateFlagBits](#), which is specified as:

```
// Flag bits for XrViewStateFlags
static const XrViewStateFlags XR_VIEW_STATE_ORIENTATION_VALID_BIT = 0x00000001;
static const XrViewStateFlags XR_VIEW_STATE_POSITION_VALID_BIT = 0x00000002;
static const XrViewStateFlags XR_VIEW_STATE_ORIENTATION_TRACKED_BIT = 0x00000004;
static const XrViewStateFlags XR_VIEW_STATE_POSITION_TRACKED_BIT = 0x00000008;
```

The flag bits have the following meanings:



## Flag Descriptions

- `XR_VIEW_STATE_ORIENTATION_VALID_BIT` indicates whether all `XrView` orientations contain valid data. Applications **must** not read any of the `XrView::pose orientation` fields if this flag is unset. `XR_VIEW_STATE_ORIENTATION_TRACKED_BIT` **should** generally remain set when this bit is set for views on a tracked headset or handheld device.
- `XR_VIEW_STATE_POSITION_VALID_BIT` indicates whether all `XrView` positions contain valid data. Applications **must** not read any of the `XrView::pose position` fields if this flag is unset. When a view loses tracking, runtimes **should** continue to provide valid but untracked view `position` values that are inferred or last-known, so long as it's still meaningful for the application to render content using that position, clearing `XR_VIEW_STATE_POSITION_TRACKED_BIT` until tracking is recovered.
- `XR_VIEW_STATE_ORIENTATION_TRACKED_BIT` indicates whether all `XrView` orientations represent an actively tracked orientation. This bit **should** generally remain set when `XR_VIEW_STATE_ORIENTATION_VALID_BIT` is set for views on a tracked headset or handheld device.
- `XR_VIEW_STATE_POSITION_TRACKED_BIT` indicates whether all `XrView` positions represent an actively tracked position. When a view loses tracking, runtimes **should** continue to provide valid but untracked view `position` values that are inferred or last-known, e.g. based on neck model updates, inertial dead reckoning, or a last-known position, so long as it's still meaningful for the application to render content using that position.

## 10.3. Frame Synchronization

An application synchronizes its rendering loop to the runtime by calling `xrWaitFrame`.

The `xrWaitFrame` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrWaitFrame(
    XrSession session,
    const XrFrameWaitInfo* frameWaitInfo,
    XrFrameState* frameState);
```

## Parameter Descriptions

- `session` is a valid [XrSession](#) handle.
- `frameWaitInfo` exists for extensibility purposes, it is `NULL` or a pointer to a valid [XrFrameWaitInfo](#).
- `frameState` is a pointer to a valid [XrFrameState](#), an output parameter.

[xrWaitFrame](#) throttles the application frame loop in order to synchronize application frame submissions with the display. [xrWaitFrame](#) returns a predicted display time for the next time that the runtime predicts a composited frame will be displayed. The runtime **may** affect this computation by changing the return values and throttling of [xrWaitFrame](#) in response to feedback from frame submission and completion times in [xrEndFrame](#). A subsequent [xrWaitFrame](#) call **must** block until the previous frame has been begun with [xrBeginFrame](#) and **must** unblock independently of the corresponding call to [xrEndFrame](#). Refer to [xrBeginSession](#) for details on how a transition to [session running](#) resets the frame function call order.

When less than one frame interval has passed since the previous return from [xrWaitFrame](#), the runtime **should** block until the beginning of the next frame interval. If more than one frame interval has passed since the last return from [xrWaitFrame](#), the runtime **may** return immediately or block until the beginning of the next frame interval.

In the case that an application has pipelined frame submissions, the application **should** compute the appropriate target display time using both the predicted display time and predicted display interval. The application **should** use the computed target display time when requesting space and view locations for rendering.

The [XrFrameState::predictedDisplayTime](#) returned by [xrWaitFrame](#) **must** be monotonically increasing.

The runtime **may** dynamically adjust the start time of the frame interval relative to the display hardware's refresh cycle to minimize graphics processor contention between the application and the compositor.

[xrWaitFrame](#) **must** be callable from any thread, including a different thread than [xrBeginFrame](#) /[xrEndFrame](#) are being called from.

Calling [xrWaitFrame](#) **must** be externally synchronized by the application, concurrent calls **may** result in undefined behavior.

The runtime **must** return `XR_ERROR_SESSION_NOT_RUNNING` if the `session` is not running.



### Note

The engine simulation **should** advance based on the display time. Every stage in the engine pipeline **should** use the exact same display time for one particular application-generated frame. An accurate and consistent display time across all stages and threads in the engine pipeline is important to avoid object motion judder. If the application has multiple pipeline stages, the application **should** pass its computed display time through its pipeline, as `xrWaitFrame` **must** be called only once per frame.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- If `frameWaitInfo` is not `NULL`, `frameWaitInfo` **must** be a pointer to a valid `XrFrameWaitInfo` structure
- `frameState` **must** be a pointer to an `XrFrameState` structure

## Thread Safety

- Access to the `session` parameter by any other `xrWaitFrame` call **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SESSION_NOT_RUNNING`

The `XrFrameWaitInfo` structure is defined as:

```
typedef struct XrFrameWaitInfo {
    XrStructureType    type;
    const void*        next;
} XrFrameWaitInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

Because this structure only exists to support extension-specific structures, [xrWaitFrame](#) **must** accept a `NULL` argument for `xrWaitFrame::frameWaitInfo` for applications that are not using any relevant extensions.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_FRAME_WAIT_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrFrameState](#) structure is defined as:

```
typedef struct XrFrameState {
    XrStructureType    type;
    void*              next;
    XrTime              predictedDisplayTime;
    XrDuration          predictedDisplayPeriod;
    XrBool32            shouldRender;
} XrFrameState;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `predictedDisplayTime` is the anticipated display [XrTime](#) for the next application-generated frame.
- `predictedDisplayPeriod` is the [XrDuration](#) of the display period for the next application-generated frame, for use in predicting display times beyond the next one.
- `shouldRender` is `XR_TRUE` if the application **should** render its layers as normal and submit them to [xrEndFrame](#). When this value is `XR_FALSE`, the application **should** avoid heavy GPU work where possible, for example by skipping layer rendering and then omitting those layers when calling [xrEndFrame](#).

[XrFrameState](#) describes the time at which the next frame will be displayed to the user. `predictedDisplayTime` **must** refer to the midpoint of the interval during which the frame is displayed. The runtime **may** report a different `predictedDisplayPeriod` from the hardware's refresh cycle.

For any frame where `shouldRender` is `XR_FALSE`, the application **should** avoid heavy GPU work for that frame, for example by not rendering its layers. This typically happens when the application is transitioning into or out of a running session, or when some system UI is fully covering the application at the moment. As long as the session **is running**, the application **should** keep running the frame loop to maintain the frame synchronization to the runtime, even if this requires calling [xrEndFrame](#) with all layers omitted.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_FRAME_STATE`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrSecondaryViewConfigurationFrameStateMSFT](#)

## 10.4. Frame Submission

Every application **must** call [xrBeginFrame](#) before calling [xrEndFrame](#), and **should** call [xrEndFrame](#) before calling [xrBeginFrame](#) again. Calling [xrEndFrame](#) again without a prior call to [xrBeginFrame](#) **must** result in `XR_ERROR_CALL_ORDER_INVALID` being returned by [xrEndFrame](#). An application **may** call [xrBeginFrame](#) again if the prior [xrEndFrame](#) fails or if the application wishes to discard an in-progress frame. A successful call to [xrBeginFrame](#) again with no intervening [xrEndFrame](#) call **must** result in the success code `XR_FRAME_DISCARDED` being returned from [xrBeginFrame](#). In this case it is assumed that the [xrBeginFrame](#) refers to the next frame and the previously begun frame is forfeited by the application.

An application **may** call `xrEndFrame` without having called `xrReleaseSwapchainImage` since the previous call to `xrEndFrame` for any swapchain passed to `xrEndFrame`. Applications **should** call `xrBeginFrame` right before executing any graphics device work for a given frame, as opposed to calling it afterwards. The runtime **must** only compose frames whose `xrBeginFrame` and `xrEndFrame` both return success codes. While `xrBeginFrame` and `xrEndFrame` do not need to be called on the same thread, the application **must** handle synchronization if they are called on separate threads.

The `xrBeginFrame` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrBeginFrame(
    XrSession          session,
    const XrFrameBeginInfo* frameBeginInfo);
```

### Parameter Descriptions

- `session` is a valid `XrSession` handle.
- `frameBeginInfo` exists for extensibility purposes, it is `NULL` or a pointer to a valid `XrFrameBeginInfo`.

`xrBeginFrame` is called prior to the start of frame rendering. The application **should** still call `xrBeginFrame` but omit rendering work for the frame if `XrFrameState::shouldRender` is `XR_FALSE`.

Runtimes **must** not perform frame synchronization or throttling through the `xrBeginFrame` function and **should** instead do so through `xrWaitFrame`.

The runtime **must** return the error code `XR_ERROR_CALL_ORDER_INVALID` if there was no corresponding successful call to `xrWaitFrame`. The runtime **must** return the success code `XR_FRAME_DISCARDED` if a prior `xrBeginFrame` has been called without an intervening call to `xrEndFrame`. Refer to `xrBeginSession` for details on how a transition to `session running` resets the frame function call order.

The runtime **must** return `XR_ERROR_SESSION_NOT_RUNNING` if the `session` is not running.

### Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- If `frameBeginInfo` is not `NULL`, `frameBeginInfo` **must** be a pointer to a valid `XrFrameBeginInfo` structure

## Thread Safety

- Access to the `session` parameter by any other `xrBeginFrame` or `xrEndFrame` call **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_FRAME_DISCARDED`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SESSION_NOT_RUNNING`
- `XR_ERROR_CALL_ORDER_INVALID`

The `XrFrameBeginInfo` structure is defined as:

```
typedef struct XrFrameBeginInfo {  
    XrStructureType    type;  
    const void*        next;  
} XrFrameBeginInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

Because this structure only exists to support extension-specific structures, `xrBeginFrame` will accept a

NULL argument for `xrBeginFrame::frameBeginInfo` for applications that are not using any relevant extensions.

### Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_FRAME_BEGIN_INFO`
- `next` **must** be NULL or a valid pointer to the [next structure in a structure chain](#)

The `xrEndFrame` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEndFrame(
    XrSession          session,
    const XrFrameEndInfo* frameEndInfo);
```

### Parameter Descriptions

- `session` is a valid [XrSession](#) handle.
- `frameEndInfo` is a pointer to a valid [XrFrameEndInfo](#).

`xrEndFrame` **may** return immediately to the application. `XrFrameEndInfo::displayTime` **should** be computed using values returned by `xrWaitFrame`. The runtime **should** be robust against variations in the timing of calls to `xrWaitFrame`, since a pipelined system may call `xrWaitFrame` on a separate thread from `xrBeginFrame` and `xrEndFrame` without any synchronization guarantees.

#### Note



An accurate predicted display time is very important to avoid black pull-in by reprojection and to reduce motion judder in case the runtime does not implement a translational reprojection. Reprojection **should** never display images before the display refresh period they were predicted for, even if they are completed early, because this will cause motion judder just the same. In other words, the better the predicted display time, the less latency experienced by the user.

Every call to `xrEndFrame` **must** be preceded by a successful call to `xrBeginFrame`. Failure to do so **must** result in `XR_ERROR_CALL_ORDER_INVALID` being returned by `xrEndFrame`. Refer to `xrBeginSession` for details on how a transition to [session running](#) resets the frame function call order. `XrFrameEndInfo` **may** reference swapchains into which the application has rendered for this frame. From each [XrSwapchain](#) only one image index is implicitly referenced per frame, the one corresponding to the last call to `xrReleaseSwapchainImage`. However, a specific swapchain (and by extension a specific



swapchain image index) **may** be referenced in [XrFrameEndInfo](#) multiple times. This **can** be used for example to render a side by side image into a single swapchain image and referencing it twice with differing image rectangles in different layers.

If no layers are provided then the display **must** be cleared.

[XR\\_ERROR\\_LAYER\\_INVALID](#) **must** be returned if an unknown, unsupported layer type, or [NULL](#) pointer is passed as one of the [XrFrameEndInfo::layers](#).

[XR\\_ERROR\\_LAYER\\_INVALID](#) **must** be returned if a layer references a swapchain that has no released swapchain image.

[XR\\_ERROR\\_LAYER\\_LIMIT\\_EXCEEDED](#) **must** be returned if [XrFrameEndInfo::layerCount](#) exceeds [XrSystemGraphicsProperties::maxLayerCount](#) or if the runtime is unable to composite the specified layers due to resource constraints.

[XR\\_ERROR\\_SWAPCHAIN\\_RECT\\_INVALID](#) **must** be returned if [XrFrameEndInfo::layers](#) contains a composition layer which references pixels outside of the associated swapchain image or if negatively sized.

[XR\\_ERROR\\_ENVIRONMENT\\_BLEND\\_MODE\\_UNSUPPORTED](#) **must** be returned if [XrFrameEndInfo::environmentBlendMode](#) is not supported.

[XR\\_ERROR\\_SESSION\\_NOT\\_RUNNING](#) **must** be returned if the [session](#) is not running.



#### Note

Applications should discard frames for which [xrEndFrame](#) returns a recoverable error over attempting to resubmit the frame with different frame parameters to provide a more consistent experience across different runtime implementations.

### Valid Usage (Implicit)

- [session](#) **must** be a valid [XrSession](#) handle
- [frameEndInfo](#) **must** be a pointer to a valid [XrFrameEndInfo](#) structure

### Thread Safety

- Access to the [session](#) parameter by any other [xrBeginFrame](#) or [xrEndFrame](#) call **must** be externally synchronized

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_TIME\_INVALID
- XR\_ERROR\_SWAPCHAIN\_RECT\_INVALID
- XR\_ERROR\_SESSION\_NOT\_RUNNING
- XR\_ERROR\_POSE\_INVALID
- XR\_ERROR\_LAYER\_LIMIT\_EXCEEDED
- XR\_ERROR\_LAYER\_INVALID
- XR\_ERROR\_ENVIRONMENT\_BLEND\_MODE\_UNSUPPORTED
- XR\_ERROR\_CALL\_ORDER\_INVALID

The `XrFrameEndInfo` structure is defined as:

```
typedef struct XrFrameEndInfo {
    XrStructureType                type;
    const void*                    next;
    XrTime                         displayTime;
    XrEnvironmentBlendMode         environmentBlendMode;
    uint32_t                       layerCount;
    const XrCompositionLayerBaseHeader* const* layers;
} XrFrameEndInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `displayTime` is the [XrTime](#) at which this frame **should** be displayed.
- `environmentBlendMode` is the [XrEnvironmentBlendMode](#) value representing the desired [environment blend mode](#) for this frame.
- `layerCount` is the number of composition layers in this frame. The maximum supported layer count is identified by [XrSystemGraphicsProperties::maxLayerCount](#). If `layerCount` is greater than the maximum supported layer count then `XR_ERROR_LAYER_LIMIT_EXCEEDED` **must** be returned.
- `layers` is a pointer to an array of [XrCompositionLayerBaseHeader](#) pointers.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_FRAME_END_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrFrameEndInfoML](#), [XrGlobalDimmerFrameEndInfoML](#), [XrLocalDimmingFrameEndInfoMETA](#), [XrSecondaryViewConfigurationFrameEndInfoMSFT](#)
- `environmentBlendMode` **must** be a valid [XrEnvironmentBlendMode](#) value
- If `layerCount` is not `0`, `layers` **must** be a pointer to an array of `layerCount` valid [XrCompositionLayerBaseHeader](#)-based structures. See also: [XrCompositionLayerCubeKHR](#), [XrCompositionLayerCylinderKHR](#), [XrCompositionLayerEquirect2KHR](#), [XrCompositionLayerEquirectKHR](#), [XrCompositionLayerPassthroughHTC](#), [XrCompositionLayerProjection](#), [XrCompositionLayerQuad](#)

All layers submitted to [xrEndFrame](#) will be presented to the primary view configuration of the running session.

## 10.5. Frame Rate

For every application-generated frame, the application **may** call [xrEndFrame](#) to submit the application-generated composition layers. In addition, the application **must** call [xrWaitFrame](#) when the application is ready to begin preparing the next set of frame layers. [xrEndFrame](#) **may** return immediately to the application, but [xrWaitFrame](#) **must** block for an amount of time that depends on throttling of the application by the runtime. The earliest the runtime will return from [xrWaitFrame](#) is when it determines that the application **should** start drawing the next frame.

## 10.6. Compositing

Composition layers are submitted by the application via the [xrEndFrame](#) call. All composition layers to be drawn **must** be submitted with every [xrEndFrame](#) call. A layer that is omitted in this call will not be drawn by the runtime layer compositor. All views associated with projection layers **must** be supplied, or [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) **must** be returned by [xrEndFrame](#).

Composition layers **must** be drawn in the same order as they are specified in via [XrFrameEndInfo](#), with the 0th layer drawn first. Layers **must** be 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.

### 10.6.1. Composition Layer Flags

[XrCompositionLayerFlags](#) specifies options for individual composition layers, and contains a bitwise-OR of zero or more of the bits defined in [XrCompositionLayerFlagBits](#).

```
typedef XrFlags64 XrCompositionLayerFlags;
```

Valid bits for [XrCompositionLayerFlags](#) are defined by [XrCompositionLayerFlagBits](#), which is specified as:

```
// Flag bits for XrCompositionLayerFlags
static const XrCompositionLayerFlags
XR_COMPOSITION_LAYER_CORRECT_CHROMATIC_ABERRATION_BIT = 0x00000001;
static const XrCompositionLayerFlags XR_COMPOSITION_LAYER_BLEND_TEXTURE_SOURCE_ALPHA_BIT
= 0x00000002;
static const XrCompositionLayerFlags XR_COMPOSITION_LAYER_UNPREMULTIPLIED_ALPHA_BIT =
0x00000004;
```

The flag bits have the following meanings:

## Flag Descriptions

- `XR_COMPOSITION_LAYER_CORRECT_CHROMATIC_ABERRATION_BIT` (*deprecated—ignored*) — Enables chromatic aberration correction when not done by default. This flag has no effect on any known conformant runtime, and is officially deprecated in OpenXR 1.1.
- `XR_COMPOSITION_LAYER_BLEND_TEXTURE_SOURCE_ALPHA_BIT` — Enables the layer texture alpha channel.
- `XR_COMPOSITION_LAYER_UNPREMULTIPLIED_ALPHA_BIT` — Indicates the texture color channels have not been premultiplied by the texture alpha channel.

### 10.6.2. Composition Layer Blending

All types of composition layers are subject to blending with other layers. Blending of layers can be controlled by layer per-textel source alpha. Layer swapchain textures may contain an alpha channel, depending on the image format. If a submitted swapchain’s texture format does not include an alpha channel or if the `XR_COMPOSITION_LAYER_BLEND_TEXTURE_SOURCE_ALPHA_BIT` is unset, then the layer alpha is initialized to one.

If the swapchain texture format color encoding is other than RGBA, it is converted to RGBA.

If the texture color channels are encoded without premultiplying by alpha, the `XR_COMPOSITION_LAYER_UNPREMULTIPLIED_ALPHA_BIT` **should** be set. The effect of this bit alters the layer color as follows:

```
LayerColor.RGB *= LayerColor.A
```

LayerColor is then clamped to a range of [0.0, 1.0].

The layer blending operation is defined as:

```
CompositeColor = LayerColor + CompositeColor * (1 - LayerColor.A)
```

Before the first layer is composited, all components of CompositeColor are initialized to zero.

### 10.6.3. Composition Layer Types

Composition layers allow an application to offload the composition of the final image to a runtime-supplied compositor. This reduces the application’s rendering complexity since details such as frame-rate interpolation and distortion correction can be performed by the runtime. The core specification defines `XrCompositionLayerProjection` and `XrCompositionLayerQuad` layer types.

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.

The quad layer type describes a possible 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.

The classes below describe the layer types in the layer composition system.

The [XrCompositionLayerBaseHeader](#) structure is defined as:

```
typedef struct XrCompositionLayerBaseHeader {
    XrStructureType      type;
    const void*         next;
    XrCompositionLayerFlags  layerFlags;
    XrSpace              space;
} XrCompositionLayerBaseHeader;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `layerFlags` is a bitmask of [XrCompositionLayerFlagBits](#) describing flags to apply to the layer.
- `space` is the [XrSpace](#) in which the layer will be kept stable over time.

All composition layer structures begin with the elements described in the [XrCompositionLayerBaseHeader](#). The [XrCompositionLayerBaseHeader](#) structure is not intended to be directly used, but forms a basis for defining current and future structures containing composition layer information. The [XrFrameEndInfo](#) structure contains an array of pointers to these polymorphic header structures. All composition layer type pointers **must** be type-castable as an [XrCompositionLayerBaseHeader](#) pointer.

## Valid Usage (Implicit)

- **type** **must** be one of the following `XrStructureType` values:  
`XR_TYPE_COMPOSITION_LAYER_CUBE_KHR`, `XR_TYPE_COMPOSITION_LAYER_CYLINDER_KHR`,  
`XR_TYPE_COMPOSITION_LAYER_EQUIRECT2_KHR`, `XR_TYPE_COMPOSITION_LAYER_EQUIRECT_KHR`,  
`XR_TYPE_COMPOSITION_LAYER_PASSTHROUGH_HTC`, `XR_TYPE_COMPOSITION_LAYER_PROJECTION`,  
`XR_TYPE_COMPOSITION_LAYER_QUAD`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also:  
`XrCompositionLayerAlphaBlendFB`, `XrCompositionLayerColorScaleBiasKHR`,  
`XrCompositionLayerDepthTestFB`, `XrCompositionLayerImageLayoutFB`,  
`XrCompositionLayerPassthroughFB`, `XrCompositionLayerSecureContentFB`,  
`XrCompositionLayerSettingsFB`
- **layerFlags** **must** be `0` or a valid combination of `XrCompositionLayerFlagBits` values
- **space** **must** be a valid `XrSpace` handle

Many composition layer structures also contain one or more references to generic layer data stored in an `XrSwapchainSubImage` structure.

The `XrSwapchainSubImage` structure is defined as:

```
typedef struct XrSwapchainSubImage {  
    XrSwapchain    swapchain;  
    XrRect2Di     imageRect;  
    uint32_t      imageArrayIndex;  
} XrSwapchainSubImage;
```

## Member Descriptions

- **swapchain** is the `XrSwapchain` to be displayed.
- **imageRect** is an `XrRect2Di` representing the valid portion of the image to use, in pixels. It also implicitly defines the transform from normalized image coordinates into pixel coordinates. The coordinate origin depends on which graphics API is being used. See the graphics API extension details for more information on the coordinate origin definition. Note that the compositor **may** bleed in pixels from outside the bounds in some cases, for instance due to mipmapping.
- **imageArrayIndex** is the image array index, with 0 meaning the first or only array element.

## Valid Usage (Implicit)

- `swapchain` **must** be a valid `XrSwapchain` handle

Runtimes **must** return `XR_ERROR_VALIDATION_FAILURE` if the `XrSwapchainSubImage::imageArrayIndex` is equal to or greater than the `XrSwapchainCreateInfo::arraySize` that the `XrSwapchainSubImage::swapchain` was created with.

## Projection Composition

The `XrCompositionLayerProjection` layer represents planar projected images rendered from the eye point of each eye using a standard perspective projection.

The `XrCompositionLayerProjection` structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrCompositionLayerProjection {
    XrStructureType           type;
    const void*               next;
    XrCompositionLayerFlags   layerFlags;
    XrSpace                    space;
    uint32_t                   viewCount;
    const XrCompositionLayerProjectionView* views;
} XrCompositionLayerProjection;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `layerFlags` is a bitmask of `XrCompositionLayerFlagBits` describing flags to apply to the layer.
- `space` is the `XrSpace` in which the `pose` of each `XrCompositionLayerProjectionView` is evaluated over time by the compositor.
- `viewCount` is the count of views in the `views` array. This **must** be equal to the number of view poses returned by `xrLocateViews`.
- `views` is the array of type `XrCompositionLayerProjectionView` containing each projection layer view.



### Note



Because a runtime may reproject the layer over time, a projection layer should specify an [XrSpace](#) in which to maximize stability of the layer content. For example, a projection layer containing world-locked content should use an [XrSpace](#) which is also world-locked, such as the [LOCAL](#) or [STAGE](#) reference spaces. In the case that the projection layer should be head-locked, such as a heads up display, the [VIEW](#) reference space would provide the highest quality layer reprojection.

### Valid Usage (Implicit)

- **type** must be [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_PROJECTION](#)
- **next** must be [NULL](#) or a valid pointer to the [next structure in a structure chain](#). See also: [XrCompositionLayerDepthTestVARJO](#), [XrCompositionLayerReprojectionInfoMSFT](#), [XrCompositionLayerReprojectionPlaneOverrideMSFT](#)
- **layerFlags** must be [0](#) or a valid combination of [XrCompositionLayerFlagBits](#) values
- **space** must be a valid [XrSpace](#) handle
- **views** must be a pointer to an array of **viewCount** valid [XrCompositionLayerProjectionView](#) structures
- The **viewCount** parameter must be greater than [0](#)

The [XrCompositionLayerProjectionView](#) structure is defined as:

```
typedef struct XrCompositionLayerProjectionView {
    XrStructureType    type;
    const void*        next;
    XrPosef             pose;
    XrFovf              fov;
    XrSwapchainSubImage subImage;
} XrCompositionLayerProjectionView;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- **pose** is an [XrPosef](#) defining the location and orientation of this projection element in the **space** of the corresponding [XrCompositionLayerProjectionView](#).
- **fov** is the [XrFovf](#) for this projection element.
- **subImage** is the image layer [XrSwapchainSubImage](#) to use. The swapchain **must** have been created with a [XrSwapchainCreateInfo::faceCount](#) of 1.

The count and order of view poses submitted with [XrCompositionLayerProjection](#) **must** be the same order as that returned by [xrLocateViews](#). The [XrCompositionLayerProjectionView::pose](#) and [XrCompositionLayerProjectionView::fov](#) **should** almost always derive from [XrView::pose](#) and [XrView::fov](#) as found in the [xrLocateViews::views](#) array. However, applications **may** submit an [XrCompositionLayerProjectionView](#) which has a different view or FOV than that from [xrLocateViews](#). In this case, the runtime will map the view and FOV to the system display appropriately. In the case that two submitted views within a single layer overlap, they **must** be composited in view array order.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_COMPOSITION_LAYER_PROJECTION_VIEW`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrCompositionLayerDepthInfoKHR](#), [XrCompositionLayerSpaceWarpInfoFB](#)
- **subImage** **must** be a valid [XrSwapchainSubImage](#) structure

## Quad Layer Composition

The [XrCompositionLayerQuad](#) structure defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrCompositionLayerQuad {
    XrStructureType      type;
    const void*          next;
    XrCompositionLayerFlags  layerFlags;
    XrSpace               space;
    XrEyeVisibility       eyeVisibility;
    XrSwapchainSubImage  subImage;
    XrPosef               pose;
    XrExtent2Df           size;
} XrCompositionLayerQuad;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `layerFlags` is a bitmask of [XrCompositionLayerFlagBits](#) describing flags to apply to the layer.
- `space` is the [XrSpace](#) in which the `pose` of the quad layer is evaluated over time.
- `eyeVisibility` is the [XrEyeVisibility](#) for this layer.
- `subImage` is the image layer [XrSwapchainSubImage](#) to use. The swapchain **must** have been created with a [XrSwapchainCreateInfo::faceCount](#) of 1.
- `pose` is an [XrPosef](#) defining the position and orientation of the quad in the reference frame of the `space`.
- `size` is the width and height of the quad in meters.

The [XrCompositionLayerQuad](#) layer is useful for user interface elements or 2D content rendered into the virtual world. The layer's [XrSwapchainSubImage::swapchain](#) image is applied to a quad in the virtual world space. Only front face of the quad surface is visible; the back face is not visible and **must** not be drawn by the runtime. A quad layer has no thickness; it is a two-dimensional object positioned and oriented in 3D space. The position of a quad refers to the center of the quad within the given [XrSpace](#). The orientation of the quad refers to the orientation of the normal vector from the front face. The size of a quad refers to the quad's size in the x-y plane of the given [XrSpace](#)'s coordinate system. A quad with a position of {0,0,0}, rotation of {0,0,0,1} (no rotation), and a size of {1,1} refers to a 1 meter x 1 meter quad centered at {0,0,0} with its front face normal vector coinciding with the +z axis.

## Valid Usage (Implicit)

- **type** must be `XR_TYPE_COMPOSITION_LAYER_QUAD`
- **next** must be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **layerFlags** must be `0` or a valid combination of [XrCompositionLayerFlagBits](#) values
- **space** must be a valid [XrSpace](#) handle
- **eyeVisibility** must be a valid [XrEyeVisibility](#) value
- **subImage** must be a valid [XrSwapchainSubImage](#) structure

The [XrEyeVisibility](#) enum selects which of the viewer's eyes to display a layer to:

```
typedef enum XrEyeVisibility {  
    XR_EYE_VISIBILITY_BOTH = 0,  
    XR_EYE_VISIBILITY_LEFT = 1,  
    XR_EYE_VISIBILITY_RIGHT = 2,  
    XR_EYE_VISIBILITY_MAX_ENUM = 0x7FFFFFFF  
} XrEyeVisibility;
```

## Enumerant Descriptions

- `XR_EYE_VISIBILITY_BOTH` displays the layer to both eyes.
- `XR_EYE_VISIBILITY_LEFT` displays the layer to the viewer's physical left eye.
- `XR_EYE_VISIBILITY_RIGHT` displays the layer to the viewer's physical right eye.

### 10.6.4. Environment Blend Mode

After the compositor has blended and flattened all layers (including any layers added by the runtime itself), it will then present this image to the system's display. The composited image will 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**. VR applications will generally choose the `XR_ENVIRONMENT_BLEND_MODE_OPAQUE` blend mode, while AR applications will generally choose either the `XR_ENVIRONMENT_BLEND_MODE_ADDITIVE` or `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` mode.

Applications select their environment blend mode each frame as part of their call to [xrEndFrame](#). The application can inspect the set of supported environment blend modes for a given system using [xrEnumerateEnvironmentBlendModes](#), and prepare their assets and rendering techniques differently based on the blend mode they choose. For example, a black shadow rendered using the

`XR_ENVIRONMENT_BLEND_MODE_ADDITIVE` blend mode will appear transparent, and so an application in that mode **may** render a glow as a grounding effect around the black shadow to ensure the shadow can be seen. Similarly, an application designed for `XR_ENVIRONMENT_BLEND_MODE_OPAQUE` or `XR_ENVIRONMENT_BLEND_MODE_ADDITIVE` rendering **may** choose to leave garbage in their alpha channel as a side effect of a rendering optimization, but this garbage would appear as visible display artifacts if the environment blend mode was instead `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND`.

Not all systems will support all environment blend modes. For example, a VR headset may not support the `XR_ENVIRONMENT_BLEND_MODE_ADDITIVE` or `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` modes unless it has video passthrough, while an AR headset with an additive display may not support the `XR_ENVIRONMENT_BLEND_MODE_OPAQUE` or `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` modes.

For devices that can support multiple environment blend modes, such as AR phones with video passthrough, the runtime **may** optimize power consumption on the device in response to the environment blend mode that the application chooses each frame. For example, if an application on a video passthrough phone knows that it is currently rendering a 360-degree background covering all screen pixels, it can submit frames with an environment blend mode of `XR_ENVIRONMENT_BLEND_MODE_OPAQUE`, saving the runtime the cost of compositing a camera-based underlay of the physical world behind the application's layers.

The `xrEnumerateEnvironmentBlendModes` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateEnvironmentBlendModes(
    XrInstance                instance,
    XrSystemId                systemId,
    XrViewConfigurationType  viewConfigurationType,
    uint32_t                  environmentBlendModeCapacityInput,
    uint32_t*                 environmentBlendModeCountOutput,
    XrEnvironmentBlendMode*  environmentBlendModes);
```

## Parameter Descriptions

- `instance` is the instance from which `systemId` was retrieved.
- `systemId` is the `XrSystemId` whose environment blend modes will be enumerated.
- `viewConfigurationType` is the `XrViewConfigurationType` to enumerate.
- `environmentBlendModeCapacityInput` is the capacity of the `environmentBlendModes` array, or 0 to indicate a request to retrieve the required capacity.
- `environmentBlendModeCountOutput` is a pointer to the count of `environmentBlendModes` written, or a pointer to the required capacity in the case that `environmentBlendModeCapacityInput` is insufficient.
- `environmentBlendModes` is a pointer to an array of `XrEnvironmentBlendMode` values, but **can** be `NULL` if `environmentBlendModeCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `environmentBlendModes` size.

Enumerates the set of environment blend modes that this runtime supports for a given view configuration of the system. Environment blend modes **should** be in order from highest to lowest runtime preference.

Runtimes **must** always return identical buffer contents from this enumeration for the given `systemId` and `viewConfigurationType` for the lifetime of the instance.

## Valid Usage (Implicit)

- `instance` **must** be a valid `XrInstance` handle
- `viewConfigurationType` **must** be a valid `XrViewConfigurationType` value
- `environmentBlendModeCountOutput` **must** be a pointer to a `uint32_t` value
- If `environmentBlendModeCapacityInput` is not 0, `environmentBlendModes` **must** be a pointer to an array of `environmentBlendModeCapacityInput` `XrEnvironmentBlendMode` values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`
- `XR_ERROR_SYSTEM_INVALID`

The possible blend modes are specified by the `XrEnvironmentBlendMode` enumeration:

```
typedef enum XrEnvironmentBlendMode {  
    XR_ENVIRONMENT_BLEND_MODE_OPAQUE = 1,  
    XR_ENVIRONMENT_BLEND_MODE_ADDITIVE = 2,  
    XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND = 3,  
    XR_ENVIRONMENT_BLEND_MODE_MAX_ENUM = 0x7FFFFFFF  
} XrEnvironmentBlendMode;
```

## Enumerant Descriptions

- **XR\_ENVIRONMENT\_BLEND\_MODE\_OPAQUE**. The composition layers will be displayed with no view of the physical world behind them. The composited image will be 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.
- **XR\_ENVIRONMENT\_BLEND\_MODE\_ADDITIVE**. The composition layers will be additively blended with the real world behind the display. The composited image will be interpreted as an RGB image, ignoring the composited alpha channel during the additive blending. This will cause black composited pixels to appear transparent. 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.
- **XR\_ENVIRONMENT\_BLEND\_MODE\_ALPHA\_BLEND**. The composition layers will be alpha-blended with the real world behind the display. The composited image will be 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.



# Chapter 11. Input and Haptics

## 11.1. Action Overview

OpenXR applications communicate with input devices using XrActions. 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. For example, instead of the application directly querying the state of the A button when interacting with a menu, an OpenXR application instead creates a `menu_select` action at startup then asks OpenXR for the state of the action.

The application recommends that the action be assigned to a specific input source on the input device for a known [interaction profile](#), but runtimes have the ability to choose a different control depending on user preference, input device availability, or any other reason. This abstraction ensures that applications can run on a wide variety of input hardware and maximize user accessibility.

Example usage:

```
XrInstance instance; // previously initialized
XrSession session; // previously initialized

// Create an action set
XrActionSetCreateInfo actionSetInfo{XR_TYPE_ACTION_SET_CREATE_INFO};
strcpy(actionSetInfo.actionSetName, "gameplay");
strcpy(actionSetInfo.localizedActionSetName, "Gameplay");
actionSetInfo.priority = 0;
XrActionSet inGameActionSet;
CHK_XR(xrCreateActionSet(instance, &actionSetInfo, &inGameActionSet));

// create a "teleport" input action
XrActionCreateInfo actioninfo{XR_TYPE_ACTION_CREATE_INFO};
strcpy(actioninfo.actionName, "teleport");
actioninfo.actionType = XR_ACTION_TYPE_BOOLEAN_INPUT;
strcpy(actioninfo.localizedActionName, "Teleport");
XrAction teleportAction;
CHK_XR(xrCreateAction(inGameActionSet, &actioninfo, &teleportAction));

// create a "player_hit" output action
XrActionCreateInfo hapticsactioninfo{XR_TYPE_ACTION_CREATE_INFO};
strcpy(hapticsactioninfo.actionName, "player_hit");
hapticsactioninfo.actionType = XR_ACTION_TYPE_VIBRATION_OUTPUT;
strcpy(hapticsactioninfo.localizedActionName, "Player hit");
XrAction hapticsAction;
CHK_XR(xrCreateAction(inGameActionSet, &hapticsactioninfo, &hapticsAction));
```

```

XrPath triggerClickPath, hapticPath;
CHK_XR(xrStringToPath(instance, "/user/hand/right/input/trigger/click",
&triggerClickPath));
CHK_XR(xrStringToPath(instance, "/user/hand/right/output/haptic", &hapticPath))

XrPath interactionProfilePath;
CHK_XR(xrStringToPath(instance, "/interaction_profiles/vendor_x/profile_x",
&interactionProfilePath));

XrActionSuggestedBinding bindings[2];
bindings[0].action = teleportAction;
bindings[0].binding = triggerClickPath;
bindings[1].action = hapticsAction;
bindings[1].binding = hapticPath;

XrInteractionProfileSuggestedBinding
suggestedBindings{XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING};
suggestedBindings.interactionProfile = interactionProfilePath;
suggestedBindings.suggestedBindings = bindings;
suggestedBindings.countSuggestedBindings = 2;
CHK_XR(xrSuggestInteractionProfileBindings(instance, &suggestedBindings));

XrSessionActionSetsAttachInfo attachInfo{XR_TYPE_SESSION_ACTION_SETS_ATTACH_INFO};
attachInfo.countActionSets = 1;
attachInfo.actionSets = &inGameActionSet;
CHK_XR(xrAttachSessionActionSets(session, &attachInfo));

// application main loop
while (1)
{
    // sync action data
    XrActiveActionSet activeActionSet{inGameActionSet, XR_NULL_PATH};
    XrActionsSyncInfo syncInfo{XR_TYPE_ACTIONS_SYNC_INFO};
    syncInfo.countActiveActionSets = 1;
    syncInfo.activeActionSets = &activeActionSet;
    CHK_XR(xrSyncActions(session, &syncInfo));

    // query input action state
    XrActionStateBoolean teleportState{XR_TYPE_ACTION_STATE_BOOLEAN};
    XrActionStateGetInfo getInfo{XR_TYPE_ACTION_STATE_GET_INFO};
    getInfo.action = teleportAction;
    CHK_XR(xrGetActionStateBoolean(session, &getInfo, &teleportState));

    if (teleportState.changedSinceLastSync && teleportState.currentState)
    {
        // fire haptics using output action
        XrHapticVibration vibration{XR_TYPE_HAPTIC_VIBRATION};
        vibration.amplitude = 0.5;
    }
}

```

```

vibration.duration = 300;
vibration.frequency = 3000;
XrHapticActionInfo hapticActionInfo{XR_TYPE_HAPTIC_ACTION_INFO};
hapticActionInfo.action = hapticsAction;
CHK_XR(xrApplyHapticFeedback(session, &hapticActionInfo, (const
XrHapticBaseHeader*)&vibration));
    }
}

```

## 11.2. Action Sets

```
XR_DEFINE_HANDLE(XrActionSet)
```

Action sets are application-defined collections of actions. They are attached to a given [XrSession](#) with a [xrAttachSessionActionSets](#) call. They are enabled or disabled by the application via [xrSyncActions](#) depending on the current application context. For example, a game may have one set of actions that apply to controlling a character and another set for navigating a menu system. When these actions are grouped into two [XrActionSet](#) handles they can be selectively enabled and disabled using a single function call.

Actions are passed a handle to their [XrActionSet](#) when they are created.

Action sets are created by calling [xrCreateActionSet](#):

The [xrCreateActionSet](#) function is defined as:

```

// Provided by XR_VERSION_1_0
XrResult xrCreateActionSet(
    XrInstance                instance,
    const XrActionSetCreateInfo* createInfo,
    XrActionSet*              actionSet);

```

## Parameter Descriptions

- `instance` is a handle to an [XrInstance](#).
- `createInfo` is a pointer to a valid [XrActionSetCreateInfo](#) structure that defines the action set being created.
- `actionSet` is a pointer to an [XrActionSet](#) where the created action set is returned.

The [xrCreateActionSet](#) function creates an action set and returns a handle to the created action set.

## Valid Usage (Implicit)

- `instance` **must** be a valid [XrInstance](#) handle
- `createInfo` **must** be a pointer to a valid [XrActionSetCreateInfo](#) structure
- `actionSet` **must** be a pointer to an [XrActionSet](#) handle

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_PATH_FORMAT_INVALID`
- `XR_ERROR_NAME_INVALID`
- `XR_ERROR_NAME_DUPLICATED`
- `XR_ERROR_LOCALIZED_NAME_INVALID`
- `XR_ERROR_LOCALIZED_NAME_DUPLICATED`

The [XrActionSetCreateInfo](#) structure is defined as:

```

typedef struct XrActionSetCreateInfo {
    XrStructureType    type;
    const void*        next;
    char               actionSetName[XR_MAX_ACTION_SET_NAME_SIZE];
    char               localizedActionSetName[XR_MAX_LOCALIZED_ACTION_SET_NAME_SIZE];
    uint32_t           priority;
} XrActionSetCreateInfo;

```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `actionSetName` is an array containing a `NULL` terminated non-empty string with the name of this action set.
- `localizedActionSetName` is an array containing a `NULL` terminated UTF-8 string that can be presented to the user as a description of the action set. This string should be presented in the system's current active locale.
- `priority` defines which action sets' actions are active on a given input source when actions on multiple active action sets are bound to the same input source. Larger priority numbers take precedence over smaller priority numbers.

When multiple actions are bound to the same input source, the `priority` of each action set determines which bindings are suppressed. Runtimes **must** ignore input sources from action sets with a lower priority number if those specific input sources are also present in active actions within a higher priority action set. If multiple action sets with the same priority are bound to the same input source and that is the highest priority number, runtimes **must** process all those bindings at the same time.

Two actions are considered to be bound to the same input source if they use the same [identifier and optional location](#) path segments, even if they have different component segments.

When runtimes are ignoring bindings because of priority, they **must** treat the binding to that input source as though they do not exist. That means the `isActive` field **must** be `XR_FALSE` when retrieving action data, and that the runtime **must** not provide any visual, haptic, or other feedback related to the binding of that action to that input source. Other actions in the same action set which are bound to input sources that do not collide are not affected and are processed as normal.

If `actionSetName` or `localizedActionSetName` are empty strings, the runtime **must** return `XR_ERROR_NAME_INVALID` or `XR_ERROR_LOCALIZED_NAME_INVALID` respectively. If `actionSetName` or `localizedActionSetName` are duplicates of the corresponding field for any existing action set in the specified instance, the runtime **must** return `XR_ERROR_NAME_DUPLICATED` or `XR_ERROR_LOCALIZED_NAME_DUPLICATED` respectively. If the conflicting action set is destroyed, the

conflicting field is no longer considered duplicated. If `actionSetName` contains characters which are not allowed in a single level of a [well-formed path string](#), the runtime **must** return `XR_ERROR_PATH_FORMAT_INVALID`.

### Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTION_SET_CREATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `actionSetName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_ACTION_SET_NAME_SIZE`
- `localizedActionSetName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_LOCALIZED_ACTION_SET_NAME_SIZE`

The `xrDestroyActionSet` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrDestroyActionSet(
    XrActionSet          actionSet);
```

### Parameter Descriptions

- `actionSet` is the action set to destroy.

Action set handles **can** be destroyed by calling `xrDestroyActionSet`. When an action set handle is destroyed, all handles of actions in that action set are also destroyed.

The implementation **must** not free underlying resources for the action set while there are other valid handles that refer to those resources. The implementation **may** release resources for an action set when all of the action spaces for actions in that action set have been destroyed. See [Action Spaces Lifetime](#) for details.

Resources for all action sets in an instance **must** be freed when the instance containing those actions sets is destroyed.

### Valid Usage (Implicit)

- `actionSet` **must** be a valid `XrActionSet` handle

## Thread Safety

- Access to `actionSet`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_HANDLE_INVALID`

## 11.3. Creating Actions

```
XR_DEFINE_HANDLE(XrAction)
```

Action handles are used to refer to individual actions when retrieving action data, creating action spaces, or sending haptic events.

The `xrCreateAction` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrCreateAction(
    XrActionSet                actionSet,
    const XrActionCreateInfo*  createInfo,
    XrAction*                  action);
```

## Parameter Descriptions

- `actionSet` is a handle to an `XrActionSet`.
- `createInfo` is a pointer to a valid `XrActionCreateInfo` structure that defines the action being created.
- `action` is a pointer to an `XrAction` where the created action is returned.

`xrCreateAction` creates an action and returns its handle.

If `actionSet` has been included in a call to `xrAttachSessionActionSets`, the implementation **must** return `XR_ERROR_ACTIONSETS_ALREADY_ATTACHED`.

### Valid Usage (Implicit)

- `actionSet` **must** be a valid `XrActionSet` handle
- `createInfo` **must** be a pointer to a valid `XrActionCreateInfo` structure
- `action` **must** be a pointer to an `XrAction` handle

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_PATH_FORMAT_INVALID`
- `XR_ERROR_NAME_INVALID`
- `XR_ERROR_NAME_DUPLICATED`
- `XR_ERROR_LOCALIZED_NAME_INVALID`
- `XR_ERROR_LOCALIZED_NAME_DUPLICATED`
- `XR_ERROR_ACTIONSETS_ALREADY_ATTACHED`

The `XrActionCreateInfo` structure is defined as:



```

typedef struct XrActionCreateInfo {
    XrStructureType    type;
    const void*        next;
    char               actionName[XR_MAX_ACTION_NAME_SIZE];
    XrActionType       actionType;
    uint32_t           countSubactionPaths;
    const XrPath*      subactionPaths;
    char               localizedActionName[XR_MAX_LOCALIZED_ACTION_NAME_SIZE];
} XrActionCreateInfo;

```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `actionName` is an array containing a `NULL` terminated string with the name of this action.
- `actionType` is the `XrActionType` of the action to be created.
- `countSubactionPaths` is the number of elements in the `subactionPaths` array. If `subactionPaths` is `NULL`, this parameter must be 0.
- `subactionPaths` is an array of `XrPath` or `NULL`. If this array is specified, it contains one or more subaction paths that the application intends to query action state for.
- `localizedActionName` is an array containing a `NULL` terminated UTF-8 string that can be presented to the user as a description of the action. This string should be in the system's current active locale.

Subaction paths are a mechanism that enables applications to use the same action name and handle on multiple devices. Applications can query action state using subaction paths that differentiate data coming from each device. This allows the runtime to group logically equivalent actions together in system UI. For instance, an application could create a single `pick_up` action with the `/user/hand/left` and `/user/hand/right` subaction paths and use the subaction paths to independently query the state of `pick_up_with_left_hand` and `pick_up_with_right_hand`.

Applications **can** create actions with or without the `subactionPaths` set to a list of paths. If this list of paths is omitted (i.e. `subactionPaths` is set to `NULL`, and `countSubactionPaths` is set to `0`), the application is opting out of filtering action results by subaction paths and any call to get action data must also omit subaction paths.

If `subactionPaths` is specified and any of the following conditions are not satisfied, the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`:

- Each path provided is one of:

- `/user/head`
- `/user/hand/left`
- `/user/hand/right`
- `/user/gamepad`
- No path appears in the list more than once

Extensions **may** append additional top level user paths to the above list.



*Note*

Earlier revisions of the spec mentioned `/user` but it could not be implemented as specified and was removed as errata.

The runtime **must** return `XR_ERROR_PATH_UNSUPPORTED` in the following circumstances:

- The application specified subaction paths at action creation and the application called `xrGetActionState*` or a haptic function with an empty subaction path array.
- The application called `xrGetActionState*` or a haptic function with a subaction path that was not specified when the action was created.

If `actionName` or `localizedActionName` are empty strings, the runtime **must** return `XR_ERROR_NAME_INVALID` or `XR_ERROR_LOCALIZED_NAME_INVALID` respectively. If `actionName` or `localizedActionName` are duplicates of the corresponding field for any existing action in the specified action set, the runtime **must** return `XR_ERROR_NAME_DUPLICATED` or `XR_ERROR_LOCALIZED_NAME_DUPLICATED` respectively. If the conflicting action is destroyed, the conflicting field is no longer considered duplicated. If `actionName` contains characters which are not allowed in a single level of a [well-formed path string](#), the runtime **must** return `XR_ERROR_PATH_FORMAT_INVALID`.

### Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTION_CREATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `actionName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_ACTION_NAME_SIZE`
- `actionType` **must** be a valid `XrActionType` value
- If `countSubactionPaths` is not `0`, `subactionPaths` **must** be a pointer to an array of `countSubactionPaths` valid `XrPath` values
- `localizedActionName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_LOCALIZED_ACTION_NAME_SIZE`

The `XrActionType` parameter takes one of the following values:

```
typedef enum XrActionType {
    XR_ACTION_TYPE_BOOLEAN_INPUT = 1,
    XR_ACTION_TYPE_FLOAT_INPUT = 2,
    XR_ACTION_TYPE_VECTOR2F_INPUT = 3,
    XR_ACTION_TYPE_POSE_INPUT = 4,
    XR_ACTION_TYPE_VIBRATION_OUTPUT = 100,
    XR_ACTION_TYPE_MAX_ENUM = 0x7FFFFFFF
} XrActionType;
```

## Enumerant Descriptions

- `XR_ACTION_TYPE_BOOLEAN_INPUT`. The action can be passed to `xrGetActionStateBoolean` to retrieve a boolean value.
- `XR_ACTION_TYPE_FLOAT_INPUT`. The action can be passed to `xrGetActionStateFloat` to retrieve a float value.
- `XR_ACTION_TYPE_VECTOR2F_INPUT`. The action can be passed to `xrGetActionStateVector2f` to retrieve a 2D float vector.
- `XR_ACTION_TYPE_POSE_INPUT`. The action can be passed to `xrCreateActionSpace` to create a space.
- `XR_ACTION_TYPE_VIBRATION_OUTPUT`. The action can be passed to `xrApplyHapticFeedback` to send a haptic event to the runtime.

The `xrDestroyAction` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrDestroyAction(
    XrAction          action);
```

## Parameter Descriptions

- `action` is the action to destroy.

Action handles **can** be destroyed by calling `xrDestroyAction`. Handles for actions that are part of an action set are automatically destroyed when the action set's handle is destroyed.

The implementation **must** not destroy the underlying resources for an action when `xrDestroyAction` is

called. Those resources are still used to make [action spaces locatable](#) and when processing action priority in [xrSyncActions](#). Destroying the action handle removes the application's access to these resources, but has no other change on actions.

Resources for all actions in an instance **must** be freed when the instance containing those actions sets is destroyed.

### Valid Usage (Implicit)

- `action` **must** be a valid [XrAction](#) handle

### Thread Safety

- Access to `action`, and any child handles, **must** be externally synchronized

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_HANDLE_INVALID`

## 11.3.1. Input Actions & Output Actions

Input actions are used to read sensors like buttons or joysticks while output actions are used for triggering haptics or motion platforms. The type of action created by [xrCreateAction](#) depends on the value of the [XrActionType](#) argument.

A given action can either be used for either input or output, but not both. Input actions are queried using one of the `xrGetActionState*` function calls, while output actions are set using the haptics calls. If either call is used with an action of the wrong type `XR_ERROR_ACTION_TYPE_MISMATCH` **must** be returned.

## 11.4. Suggested Bindings

Applications suggest bindings for their actions to runtimes so that raw input data is mapped appropriately to the application's actions. Suggested bindings also serve as a signal indicating the hardware that has been tested by the application developer. Applications **can** suggest bindings by calling [xrSuggestInteractionProfileBindings](#) for each [interaction profile](#) that the application is developed and tested with. If bindings are provided for an appropriate interaction profile, the runtime **may** select one and input will begin to flow. Interaction profile selection changes **must** only happen

when `xrSyncActions` is called. Applications **can** call `xrGetCurrentInteractionProfile` during on a running session to learn what the active interaction profile are for a top level user path. If this value ever changes, the runtime **must** send an `XR_TYPE_EVENT_DATA_INTERACTION_PROFILE_CHANGED` event to the application to indicate that the value should be queried again.

The bindings suggested by this system are only a hint to the runtime. Some runtimes **may** choose to use a different device binding depending on user preference, accessibility settings, or for any other reason. If the runtime is using the values provided by suggested bindings, it **must** make a best effort to convert the input value to the created action and apply certain rules to that use so that suggested bindings function in the same way across runtimes. If an input value cannot be converted to the type of the action, the value **must** be ignored and not contribute to the state of the action.

For actions created with `XR_ACTION_TYPE_BOOLEAN_INPUT` when the runtime is obeying suggested bindings: Boolean input sources **must** be bound directly to the action. If the path is to a scalar value, a threshold **must** be applied to the value and values over that threshold will be `XR_TRUE`. The runtime **should** use hysteresis when applying this threshold. The threshold and hysteresis range **may** vary from device to device or component to component and are left as an implementation detail. If the path refers to the parent of input values instead of to an input value itself, the runtime **must** use `.../example/path/click` instead of `.../example/path` if it is available. If a parent path does not have a `.../click` subpath, the runtime **must** use `.../value` and apply the same thresholding that would be applied to any scalar input. In any other situation the runtime **may** provide an alternate binding for the action or it will be unbound.

For actions created with `XR_ACTION_TYPE_FLOAT_INPUT` when the runtime is obeying suggested bindings: If the input value specified by the path is scalar, the input value **must** be bound directly to the float. If the path refers to the parent of input values instead of to an input value itself, the runtime **must** use `.../example/path/value` instead of `.../example/path` as the source of the value. If a parent path does not have a `.../value` subpath, the runtime **must** use `.../click`. If the input value is boolean, the runtime **must** supply 0.0 or 1.0 as a conversion of the boolean value. In any other situation, the runtime **may** provide an alternate binding for the action or it will be unbound.

For actions created with `XR_ACTION_TYPE_VECTOR2F_INPUT` when the runtime is obeying suggested bindings: The suggested binding path **must** refer to the parent of input values instead of to the input values themselves, and that parent path **must** contain subpaths `.../x` and `.../y`. `.../x` and `.../y` **must** be bound to 'x' and 'y' of the vector, respectively. In any other situation, the runtime **may** provide an alternate binding for the action or it will be unbound.

For actions created with `XR_ACTION_TYPE_POSE_INPUT` when the runtime is obeying suggested bindings: Pose input sources **must** be bound directly to the action. If the path refers to the parent of input values instead of to an input value itself, the runtime **must** use `.../example/path/pose` instead of `.../example/path` if it is available. In any other situation the runtime **may** provide an alternate binding for the action or it will be unbound.

The `xrSuggestInteractionProfileBindings` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrSuggestInteractionProfileBindings(
    XrInstance instance,
    const XrInteractionProfileSuggestedBinding* suggestedBindings);
```

## Parameter Descriptions

- **instance** is the [XrInstance](#) for which the application would like to set suggested bindings
- **suggestedBindings** is the [XrInteractionProfileSuggestedBinding](#) that the application would like to set

The [xrSuggestInteractionProfileBindings](#) function provides action bindings for a single interaction profile. The application **can** call [xrSuggestInteractionProfileBindings](#) once per interaction profile that it supports.

The application **can** provide any number of bindings for each action.

If the application successfully calls [xrSuggestInteractionProfileBindings](#) more than once for an interaction profile, the runtime **must** discard the previous suggested bindings and replace them with the new suggested bindings for that profile.

If the interaction profile path does not follow the structure defined in [Interaction Profiles](#) or suggested bindings contain paths that do not follow the format defined in [Input subpaths](#) (further described in [XrActionSuggestedBinding](#)), the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`. If the interaction profile or input source for any of the suggested bindings does not exist in the allowlist defined in [Interaction Profile Paths](#), the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`. A runtime **must** accept every valid binding in the allowlist though it is free to ignore any of them.

If the action set for any action referenced in the **suggestedBindings** parameter has been included in a call to [xrAttachSessionActionSets](#), the implementation **must** return `XR_ERROR_ACTIONSETS_ALREADY_ATTACHED`.

## Valid Usage (Implicit)

- **instance** **must** be a valid [XrInstance](#) handle
- **suggestedBindings** **must** be a pointer to a valid [XrInteractionProfileSuggestedBinding](#) structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTIONSETS_ALREADY_ATTACHED`

The `XrInteractionProfileSuggestedBinding` structure is defined as:

```
typedef struct XrInteractionProfileSuggestedBinding {
    XrStructureType          type;
    const void*              next;
    XrPath                   interactionProfile;
    uint32_t                 countSuggestedBindings;
    const XrActionSuggestedBinding* suggestedBindings;
} XrInteractionProfileSuggestedBinding;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `interactionProfile` is the `XrPath` of an interaction profile.
- `countSuggestedBindings` is the number of suggested bindings in the array pointed to by `suggestedBindings`.
- `suggestedBindings` is a pointer to an array of `XrActionSuggestedBinding` structures that define all of the application's suggested bindings for the specified interaction profile.

## Valid Usage (Implicit)

- **type** must be `XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING`
- **next** must be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrBindingModificationsKHR](#)
- **suggestedBindings** must be a pointer to an array of `countSuggestedBindings` valid [XrActionSuggestedBinding](#) structures
- The `countSuggestedBindings` parameter must be greater than 0

The [XrActionSuggestedBinding](#) structure is defined as:

```
typedef struct XrActionSuggestedBinding {  
    XrAction    action;  
    XrPath      binding;  
} XrActionSuggestedBinding;
```

## Member Descriptions

- **action** is the [XrAction](#) handle for an action
- **binding** is the [XrPath](#) of a binding for the action specified in **action**. This path is any top level user path plus input source path, for example `/user/hand/right/input/trigger/click`. See [suggested bindings](#) for more details.

## Valid Usage (Implicit)

- **action** must be a valid [XrAction](#) handle

The [xrAttachSessionActionSets](#) function is defined as:

```
// Provided by XR_VERSION_1_0  
XrResult xrAttachSessionActionSets(  
    XrSession          session,  
    const XrSessionActionSetsAttachInfo* attachInfo);
```



## Parameter Descriptions

- `session` is the [XrSession](#) to attach the action sets to.
- `attachInfo` is the [XrSessionActionSetsAttachInfo](#) to provide information to attach action sets to the session.

`xrAttachSessionActionSets` attaches the [XrActionSet](#) handles in [XrSessionActionSetsAttachInfo::actionSets](#) to the `session`. Action sets **must** be attached in order to be synchronized with [xrSyncActions](#).

When an action set is attached to a session, that action set becomes immutable. See [xrCreateAction](#) and [xrSuggestInteractionProfileBindings](#) for details.

After action sets are attached to a session, if any unattached actions are passed to functions for the same session, then for those functions the runtime **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED`.

The runtime **must** return `XR_ERROR_ACTIONSETS_ALREADY_ATTACHED` if `xrAttachSessionActionSets` is called more than once for a given `session`.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `attachInfo` **must** be a pointer to a valid [XrSessionActionSetsAttachInfo](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_ACTIONSETS_ALREADY_ATTACHED`

The [XrSessionActionSetsAttachInfo](#) structure is defined as:

```
typedef struct XrSessionActionSetsAttachInfo {
    XrStructureType    type;
    const void*        next;
    uint32_t           countActionSets;
    const XrActionSet* actionSets;
} XrSessionActionSetsAttachInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `countActionSets` is an integer specifying the number of valid elements in the `actionSets` array.
- `actionSets` is a pointer to an array of one or more [XrActionSet](#) handles to be attached to the session.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_SESSION_ACTION_SETS_ATTACH_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `actionSets` **must** be a pointer to an array of `countActionSets` valid [XrActionSet](#) handles
- The `countActionSets` parameter **must** be greater than 0

## 11.5. Current Interaction Profile

The [xrGetCurrentInteractionProfile](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetCurrentInteractionProfile(
    XrSession          session,
    XrPath              topLevelUserPath,
    XrInteractionProfileState* interactionProfile);
```

## Parameter Descriptions

- `session` is the [XrSession](#) for which the application would like to retrieve the current interaction profile.
- `topLevelUserPath` is the top level user path the application would like to retrieve the interaction profile for.
- `interactionProfile` is a pointer to an [XrInteractionProfileState](#) structure to receive the current interaction profile.

[xrGetCurrentInteractionProfile](#) retrieves the current interaction profile for a top level user path.

The runtime **must** return only interaction profiles for which the application has provided suggested bindings with [xrSuggestInteractionProfileBindings](#) or `XR_NULL_PATH`. The runtime **may** return interaction profiles that do not represent physically present hardware, for example if the runtime is using a known interaction profile to bind to hardware that the application is not aware of. The runtime **may** return the last-known interaction profile in the event that no controllers are active.

If [xrAttachSessionActionSets](#) has not yet been called for the `session`, the runtime **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED`. If `topLevelUserPath` is not one of the top level user paths described in [/user paths](#), the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `interactionProfile` **must** be a pointer to an [XrInteractionProfileState](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The `XrInteractionProfileState` structure is defined as:

```
typedef struct XrInteractionProfileState {  
    XrStructureType    type;  
    void*              next;  
    XrPath              interactionProfile;  
} XrInteractionProfileState;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `interactionProfile` is the `XrPath` of the interaction profile path for the `xrGetCurrentInteractionProfile::topLevelUserPath` used to retrieve this state, or `XR_NULL_PATH` if there is no active interaction profile at that top level user path.

The runtime **must** only include interaction profiles that the application has provided bindings for via `xrSuggestInteractionProfileBindings` or `XR_NULL_PATH`. If the runtime is rebinding an interaction profile provided by the application to a device that the application did not provide bindings for, it

**must** return the interaction profile path that it is emulating. If the runtime is unable to provide input because it cannot emulate any of the application-provided interaction profiles, it **must** return [XR\\_NULL\\_PATH](#).

### Valid Usage (Implicit)

- **type** **must** be [XR\\_TYPE\\_INTERACTION\\_PROFILE\\_STATE](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataInteractionProfileChanged](#) structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrEventDataInteractionProfileChanged {
    XrStructureType    type;
    const void*        next;
    XrSession          session;
} XrEventDataInteractionProfileChanged;
```

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is [NULL](#) or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- **session** is the [XrSession](#) for which at least one of the interaction profiles for a top level path has changed.

The [XrEventDataInteractionProfileChanged](#) event is queued to notify the application that the current interaction profile for one or more top level user paths has changed. This event **must** only be sent for interaction profiles that the application indicated its support for via [xrSuggestInteractionProfileBindings](#). This event **must** only be queued for running sessions.

Upon receiving this event, an application **can** call [xrGetCurrentInteractionProfile](#) for each top level user path in use, if its behavior depends on the current interaction profile.

### Valid Usage (Implicit)

- **type** **must** be [XR\\_TYPE\\_EVENT\\_DATA\\_INTERACTION\\_PROFILE\\_CHANGED](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

## 11.6. Reading Input Action State

The current state of an input action can be obtained by calling the `xrGetActionState*` function call that matches the `XrActionType` provided when the action was created. If a mismatched call is used to retrieve the state `XR_ERROR_ACTION_TYPE_MISMATCH` **must** be returned. `xrGetActionState*` calls for an action in an action set never bound to the session with `xrAttachSessionActionSets` **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED`.

The result of calls to `xrGetActionState*` for an `XrAction` and subaction path **must** not change between calls to `xrSyncActions`. When the combination of the parent `XrActionSet` and subaction path for an action is passed to `xrSyncActions`, the runtime **must** update the results from `xrGetActionState*` after this call with any changes to the state of the underlying hardware. When the parent action set and subaction path for an action is removed from or added to the list of active action sets passed to `xrSyncActions`, the runtime **must** update `isActive` to reflect the new active state after this call. In all cases the runtime **must** not change the results of `xrGetActionState*` calls between calls to `xrSyncActions`.

When `xrGetActionState*` or haptic output functions are called while the session is **not focused**, the runtime **must** set the `isActive` value to `XR_FALSE` and suppress all haptic output. Furthermore, the runtime should stop all in-progress haptic events when a session loses focus.

When retrieving action state, `lastChangeTime` **must** be set to the runtime's best estimate of when the physical state of the part of the device bound to that action last changed.

The `currentState` value is computed based on the current sync, combining the underlying input sources bound to the provided `subactionPaths` within this action.

The `changedSinceLastSync` value **must** be `XR_TRUE` if the computed `currentState` value differs from the `currentState` value that would have been computed as of the previous sync for the same `subactionPaths`. If there is no previous sync, or the action was not active for the previous sync, the `changedSinceLastSync` value **must** be set to `XR_FALSE`.

The `isActive` value **must** be `XR_TRUE` whenever an action is bound and a source is providing state data for the current sync. If the action is unbound or no source is present, the `isActive` value **must** be `XR_FALSE`. For any action which is inactive, the runtime **must** return zero (or `XR_FALSE`) for state, `XR_FALSE` for `changedSinceLastSync`, and `0` for `lastChangeTime`.

### 11.6.1. Resolving a single action bound to multiple inputs or outputs

It is often the case that a single action will be bound to multiple physical inputs simultaneously. In these circumstances, the runtime **must** resolve the ambiguity in that multiple binding as follows:

The current state value is selected based on the type of the action:

- Boolean actions - The current state **must** be the result of a boolean **OR** of all bound inputs
- Float actions - The current state **must** be the state of the input with the largest absolute value

- Vector2 actions - The current state **must** be the state of the input with the longest length
- Pose actions - The current state **must** be the state of a single pose source. The source of the pose **must** only be changed during a call to `xrSyncAction`. The runtime **should** only change the source in response to user actions, such as picking up a new controller, or external events, such as a controller running out of battery.
- Haptic actions - The runtime **must** send output events to all bound haptic devices

## 11.6.2. Structs to describe action and subaction paths

The `XrActionStateGetInfo` structure is used to provide action and subaction paths when calling `xrGetActionState*` function. It is defined as:

```
typedef struct XrActionStateGetInfo {
    XrStructureType    type;
    const void*        next;
    XrAction            action;
    XrPath              subactionPath;
} XrActionStateGetInfo;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `action` is the `XrAction` being queried.
- `subactionPath` is the subaction path `XrPath` to query data from, or `XR_NULL_PATH` to specify all subaction paths. If the subaction path is specified, it is one of the subaction paths that were specified when the action was created. If the subaction path was not specified when the action was created, the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`. If this parameter is specified, the runtime **must** return data that originates only from the subaction paths specified.

See `XrActionCreateInfo` for a description of subaction paths, and the restrictions on their use.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_ACTION_STATE_GET_INFO`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **action** **must** be a valid [XrAction](#) handle

The [XrHapticActionInfo](#) structure is used to provide action and subaction paths when calling `xrHapticFeedback` function. It is defined as:

```
typedef struct XrHapticActionInfo {  
    XrStructureType    type;  
    const void*        next;  
    XrAction            action;  
    XrPath              subactionPath;  
} XrHapticActionInfo;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- **action** is the [XrAction](#) handle for the desired output haptic action.
- **subactionPath** is the subaction path [XrPath](#) of the device to send the haptic event to, or [XR\\_NULL\\_PATH](#) to specify all subaction paths. If the subaction path is specified, it is one of the subaction paths that were specified when the action was created. If the subaction path was not specified when the action was created, the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`. If this parameter is specified, the runtime **must** trigger the haptic events only on the device from the subaction path.

See [XrActionCreateInfo](#) for a description of subaction paths, and the restrictions on their use.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_HAPTIC_ACTION_INFO`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **action** **must** be a valid [XrAction](#) handle



### 11.6.3. Boolean Actions

`xrGetActionStateBoolean` retrieves the current state of a boolean action. It is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetActionStateBoolean(
    XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStateBoolean* state);
```

#### Parameter Descriptions

- `session` is the `XrSession` to query.
- `getInfo` is a pointer to `XrActionStateGetInfo` to provide action and subaction paths information.
- `state` is a pointer to a valid `XrActionStateBoolean` into which the state will be placed.

#### Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `getInfo` **must** be a pointer to a valid `XrActionStateGetInfo` structure
- `state` **must** be a pointer to an `XrActionStateBoolean` structure

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_PATH\_UNSUPPORTED
- XR\_ERROR\_PATH\_INVALID
- XR\_ERROR\_ACTION\_TYPE\_MISMATCH
- XR\_ERROR\_ACTIONSET\_NOT\_ATTACHED

The `XrActionStateBoolean` structure is defined as:

```
typedef struct XrActionStateBoolean {
    XrStructureType    type;
    void*              next;
    XrBool32           currentState;
    XrBool32           changedSinceLastSync;
    XrTime              lastChangeTime;
    XrBool32           isActive;
} XrActionStateBoolean;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `currentState` is the current state of the action.
- `changedSinceLastSync` is `XR_TRUE` if the value of `currentState` is different than it was before the most recent call to [xrSyncActions](#). This parameter can be combined with `currentState` to detect rising and falling edges since the previous call to [xrSyncActions](#). E.g. if both `changedSinceLastSync` and `currentState` are `XR_TRUE` then a rising edge (`XR_FALSE` to `XR_TRUE`) has taken place.
- `lastChangeTime` is the [XrTime](#) associated with the most recent change to this action's state.
- `isActive` is `XR_TRUE` if and only if there exists an input source that is contributing to the current state of this action.

When multiple input sources are bound to this action, the `currentState` follows [the previously defined rule to resolve ambiguity](#).

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTION_STATE_BOOLEAN`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 11.6.4. Scalar and Vector Actions

[xrGetActionStateFloat](#) retrieves the current state of a floating-point action. It is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetActionStateFloat(
    XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStateFloat* state);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to query.
- `getInfo` is a pointer to [XrActionStateGetInfo](#) to provide action and subaction paths information.
- `state` is a pointer to a valid [XrActionStateFloat](#) into which the state will be placed.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `getInfo` **must** be a pointer to a valid [XrActionStateGetInfo](#) structure
- `state` **must** be a pointer to an [XrActionStateFloat](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The [XrActionStateFloat](#) structure is defined as:

```
typedef struct XrActionStateFloat {
    XrStructureType    type;
    void*              next;
    float              currentState;
    XrBool32           changedSinceLastSync;
    XrTime             lastChangeTime;
    XrBool32           isActive;
} XrActionStateFloat;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `currentState` is the current state of the Action.
- `changedSinceLastSync` is `XR_TRUE` if the value of `currentState` is different than it was before the most recent call to [xrSyncActions](#).
- `lastChangeTime` is the [XrTime](#) associated with the most recent change to this action's state.
- `isActive` is `XR_TRUE` if and only if there exists an input source that is contributing to the current state of this action.

When multiple input sources are bound to this action, the `currentState` follows [the previously defined rule to resolve ambiguity](#).

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTION_STATE_FLOAT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

[xrGetActionStateVector2f](#) retrieves the current state of a two-dimensional vector action. It is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetActionStateVector2f(
    XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStateVector2f* state);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to query.
- `getInfo` is a pointer to [XrActionStateGetInfo](#) to provide action and subaction paths information.
- `state` is a pointer to a valid [XrActionStateVector2f](#) into which the state will be placed.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `getInfo` **must** be a pointer to a valid [XrActionStateGetInfo](#) structure
- `state` **must** be a pointer to an [XrActionStateVector2f](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The [XrActionStateVector2f](#) structure is defined as:

```
typedef struct XrActionStateVector2f {
    XrStructureType    type;
    void*              next;
    XrVector2f         currentState;
    XrBool32           changedSinceLastSync;
    XrTime             lastChangeTime;
    XrBool32           isActive;
} XrActionStateVector2f;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `currentState` is the current [XrVector2f](#) state of the Action.
- `changedSinceLastSync` is `XR_TRUE` if the value of `currentState` is different than it was before the most recent call to [xrSyncActions](#).
- `lastChangeTime` is the [XrTime](#) associated with the most recent change to this action's state.
- `isActive` is `XR_TRUE` if and only if there exists an input source that is contributing to the current state of this action.

When multiple input sources are bound to this action, the `currentState` follows [the previously defined rule to resolve ambiguity](#).

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTION_STATE_VECTOR2F`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 11.6.5. Pose Actions

The [xrGetActionStatePose](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetActionStatePose(
    XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStatePose* state);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to query.
- `getInfo` is a pointer to [XrActionStateGetInfo](#) to provide action and subaction paths information.
- `state` is a pointer to a valid [XrActionStatePose](#) into which the state will be placed.

[xrGetActionStatePose](#) returns information about the binding and active state for the specified action. To determine the pose of this action at a historical or predicted time, the application **can** create an action space using [xrCreateActionSpace](#). Then, after each sync, the application **can** locate the pose of this action space within a base space using [xrLocateSpace](#).

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `getInfo` **must** be a pointer to a valid [XrActionStateGetInfo](#) structure
- `state` **must** be a pointer to an [XrActionStatePose](#) structure



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The `XrActionStatePose` structure is defined as:

```
typedef struct XrActionStatePose {  
    XrStructureType    type;  
    void*              next;  
    XrBool32           isActive;  
} XrActionStatePose;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `isActive` is `XR_TRUE` if and only if there exists an input source that is being tracked by this pose action.

A pose action **must** not be bound to multiple input sources, according to [the previously defined rule](#).

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_ACTION_STATE_POSE`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 11.7. Output Actions and Haptics

Haptic feedback is sent to a device using the [xrApplyHapticFeedback](#) function. The `hapticEvent` points to a supported event structure. All event structures have in common that the first element is an [XrHapticBaseHeader](#) which can be used to determine the type of the haptic event.

Haptic feedback may be immediately halted for a haptic action using the [xrStopHapticFeedback](#) function.

Output action requests activate immediately and **must** not wait for the next call to [xrSyncActions](#).

If a haptic event is sent to an action before a previous haptic event completes, the latest event will take precedence and the runtime **must** cancel all preceding incomplete haptic events on that action.

Output action requests **must** be discarded and have no effect on hardware if the application's session is not focused.

Output action requests for an action in an action set never attached to the session with [xrAttachSessionActionSets](#) **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED`.

The only haptics type supported by unextended OpenXR is [XrHapticVibration](#).

The [xrApplyHapticFeedback](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrApplyHapticFeedback(
    XrSession session,
    const XrHapticActionInfo* hapticActionInfo,
    const XrHapticBaseHeader* hapticFeedback);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to start outputting to.
- `hapticActionInfo` is a pointer to [XrHapticActionInfo](#) to provide action and subaction paths information.
- `hapticFeedback` is a pointer to a haptic event structure which starts with an [XrHapticBaseHeader](#).

Triggers a haptic event through the specified action of type `XR_ACTION_TYPE_VIBRATION_OUTPUT`. The runtime **should** deliver this request to the appropriate device, but exactly which device, if any, this event is sent to is up to the runtime to decide. If an appropriate device is unavailable the runtime **may** ignore this request for haptic feedback.

If `session` is not focused, the runtime **must** return `XR_SESSION_NOT_FOCUSED`, and not trigger a haptic event.

If another haptic event from this session is currently happening on the device bound to this action, the runtime **must** interrupt that other event and replace it with the new one.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `hapticActionInfo` **must** be a pointer to a valid [XrHapticActionInfo](#) structure
- `hapticFeedback` **must** be a pointer to a valid [XrHapticBaseHeader](#)-based structure. See also: [XrHapticAmplitudeEnvelopeVibrationFB](#), [XrHapticPcmVibrationFB](#), [XrHapticVibration](#)

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_SESSION_NOT_FOCUSED`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The `XrHapticBaseHeader` structure is defined as:

```
typedef struct XrHapticBaseHeader {  
    XrStructureType    type;  
    const void*       next;  
} XrHapticBaseHeader;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure. This base structure itself has no associated `XrStructureType` value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.

## Valid Usage (Implicit)

- `type` **must** be one of the following `XrStructureType` values:  
`XR_TYPE_HAPTIC_AMPLITUDE_ENVELOPE_VIBRATION_FB`, `XR_TYPE_HAPTIC_PCM_VIBRATION_FB`,  
`XR_TYPE_HAPTIC_VIBRATION`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrHapticVibration` structure is defined as:

```
// Provided by XR_VERSION_1_0
typedef struct XrHapticVibration {
    XrStructureType    type;
    const void*        next;
    XrDuration          duration;
    float               frequency;
    float               amplitude;
} XrHapticVibration;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `duration` is the number of nanoseconds the vibration **should** last. If [XR\\_MIN\\_HAPTIC\\_DURATION](#) is specified, the runtime **must** produce a short haptics pulse of minimal supported duration for the haptic device.
- `frequency` is the frequency of the vibration in Hz. If [XR\\_FREQUENCY\\_UNSPECIFIED](#) is specified, it is left to the runtime to decide the optimal frequency value to use.
- `amplitude` is the amplitude of the vibration between 0.0 and 1.0.

The `XrHapticVibration` is used in calls to [xrApplyHapticFeedback](#) that trigger `vibration` output actions.

The `duration`, and `frequency` parameters **may** be clamped to implementation-dependent ranges.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_HAPTIC_VIBRATION`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

`XR_MIN_HAPTIC_DURATION` is used to indicate to the runtime that a short haptic pulse of the minimal supported duration for the haptic device.

```
// Provided by XR_VERSION_1_0
#define XR_MIN_HAPTIC_DURATION -1
```

`XR_FREQUENCY_UNSPECIFIED` is used to indicate that the application wants the runtime to decide what the optimal frequency is for the haptic pulse.

```
// Provided by XR_VERSION_1_0
#define XR_FREQUENCY_UNSPECIFIED 0
```

The `xrStopHapticFeedback` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrStopHapticFeedback(
    XrSession session,
    const XrHapticActionInfo* hapticActionInfo);
```

## Parameter Descriptions

- **session** is the [XrSession](#) to stop outputting to.
- **hapticActionInfo** is a pointer to an [XrHapticActionInfo](#) to provide action and subaction path information.

If a haptic event from this [XrAction](#) is in progress, when this function is called the runtime **must** stop that event.

If **session** is not focused, the runtime **must** return `XR_SESSION_NOT_FOCUSED`.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `hapticActionInfo` **must** be a pointer to a valid `XrHapticActionInfo` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_SESSION_NOT_FOCUSED`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

## 11.8. Input Action State Synchronization

The `xrSyncActions` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrSyncActions(
    XrSession session,
    const XrActionsSyncInfo* syncInfo);
```

## Parameter Descriptions

- `session` is a handle to the [XrSession](#) that all provided action set handles belong to.
- `syncInfo` is an [XrActionsSyncInfo](#) providing information to synchronize action states.

`xrSyncActions` updates the current state of input actions. Repeated input action state queries between subsequent synchronization calls **must** return the same values. The [XrActionSet](#) structures referenced in the [XrActionsSyncInfo::activeActionSets](#) **must** have been previously attached to the session via [xrAttachSessionActionSets](#). If any action sets not attached to this session are passed to `xrSyncActions` it **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED`. Subsets of the bound action sets **can** be synchronized in order to control which actions are seen as active.

If `session` is not focused, the runtime **must** return `XR_SESSION_NOT_FOCUSED`, and all action states in the session **must** be inactive.

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `syncInfo` **must** be a pointer to a valid [XrActionsSyncInfo](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_SESSION_NOT_FOCUSED`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`



The [XrActionsSyncInfo](#) structure is defined as:

```
typedef struct XrActionsSyncInfo {
    XrStructureType    type;
    const void*        next;
    uint32_t           countActiveActionSets;
    const XrActiveActionSet* activeActionSets;
} XrActionsSyncInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `countActiveActionSets` is an integer specifying the number of valid elements in the `activeActionSets` array.
- `activeActionSets` is `NULL` or a pointer to an array of one or more [XrActiveActionSet](#) structures that should be synchronized.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_ACTIONS_SYNC_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrActiveActionSetPrioritiesEXT](#)
- If `countActiveActionSets` is not 0, `activeActionSets` **must** be a pointer to an array of `countActiveActionSets` valid [XrActiveActionSet](#) structures

The [XrActiveActionSet](#) structure is defined as:

```
typedef struct XrActiveActionSet {
    XrActionSet    actionSet;
    XrPath         subactionPath;
} XrActiveActionSet;
```

## Member Descriptions

- `actionSet` is the handle of the action set to activate.
- `subactionPath` is a subaction path that was declared when one or more actions in the action set was created or `XR_NULL_PATH`. If the application wants to activate the action set on more than one subaction path, it **can** include additional `XrActiveActionSet` structs with the other `subactionPath` values. Using `XR_NULL_PATH` as the value for `subactionPath`, acts as a wildcard for all subaction paths on the actions in the action set. If the subaction path was not specified on any of the actions in the actionSet when that action was created, the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`.

This structure defines a single active action set and subaction path combination. Applications **can** provide a list of these structures to the `xrSyncActions` function.

## Valid Usage (Implicit)

- `actionSet` **must** be a valid `XrActionSet` handle

## 11.9. Bound Sources

An application **can** use the `xrEnumerateBoundSourcesForAction` and `xrGetInputSourceLocalizedName` calls to prompt the user which physical inputs to use in order to perform an action. The bound **sources** are `XrPath` semantic paths representing the physical controls that an action is bound to. An action **may** be bound to multiple sources at one time, for example an action named `hold` could be bound to both the X and A buttons.

Once the bound sources for an action are obtained, the application **can** gather additional information about it. `xrGetInputSourceLocalizedName` returns a localized human-readable string describing the bound physical control, e.g. 'A Button'.

The `xrEnumerateBoundSourcesForAction` function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrEnumerateBoundSourcesForAction(
    XrSession session,
    const XrBoundSourcesForActionEnumerateInfo* enumerateInfo,
    uint32_t sourceCapacityInput,
    uint32_t* sourceCountOutput,
    XrPath* sources);
```

## Parameter Descriptions

- `session` is the [XrSession](#) being queried.
- `enumerateInfo` is an [XrBoundSourcesForActionEnumerateInfo](#) providing the query information.
- `sourceCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `sourceCountOutput` is a pointer to the count of `sources`, or a pointer to the required capacity in the case that `sourceCapacityInput` is insufficient.
- `sources` is a pointer to an application-allocated array that will be filled with the [XrPath](#) values for all bound sources. It **can** be `NULL` if `sourceCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `sources` size.

If an action is unbound, [xrEnumerateBoundSourcesForAction](#) **must** assign 0 to the value pointed-to by `sourceCountOutput` and not modify the array.

[xrEnumerateBoundSourcesForAction](#) **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED` if passed an action in an action set never attached to the session with [xrAttachSessionActionSets](#).

As bindings for actions do not change between calls to [xrSyncActions](#), [xrEnumerateBoundSourcesForAction](#) **must** enumerate the same set of bound sources, or absence of bound sources, for a given query (defined by the `enumerateInfo` parameter) between any two calls to [xrSyncActions](#).

### Note



The [XrPath](#) bound sources returned by the runtime are opaque values and **should** not be inspected or persisted. They are only intended for use in conjunction with [xrGetInputSourceLocalizedName](#).

## Valid Usage (Implicit)

- `session` **must** be a valid [XrSession](#) handle
- `enumerateInfo` **must** be a pointer to a valid [XrBoundSourcesForActionEnumerateInfo](#) structure
- `sourceCountOutput` **must** be a pointer to a `uint32_t` value
- If `sourceCapacityInput` is not 0, `sources` **must** be a pointer to an array of `sourceCapacityInput` [XrPath](#) values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The `XrBoundSourcesForActionEnumerateInfo` structure is defined as:

```
typedef struct XrBoundSourcesForActionEnumerateInfo {  
    XrStructureType    type;  
    const void*        next;  
    XrAction           action;  
} XrBoundSourcesForActionEnumerateInfo;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `action` is the handle of the action to query.

## Valid Usage (Implicit)

- **type** **must** be `XR_TYPE_BOUND_SOURCES_FOR_ACTION_ENUMERATE_INFO`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **action** **must** be a valid [XrAction](#) handle

The [xrGetInputSourceLocalizedName](#) function is defined as:

```
// Provided by XR_VERSION_1_0
XrResult xrGetInputSourceLocalizedName(
    XrSession session,
    const XrInputSourceLocalizedNameGetInfo* getInfo,
    uint32_t bufferCapacityInput,
    uint32_t* bufferCountOutput,
    char* buffer);
```

## Parameter Descriptions

- **session** is a handle to the [XrSession](#) associated with the action that reported this bound source.
- **getInfo** is an [XrInputSourceLocalizedNameGetInfo](#) providing the query information.
- **bufferCapacityInput** is the capacity of the **buffer**, or 0 to indicate a request to retrieve the required capacity.
- **bufferCountOutput** is a pointer to the count of name characters written to **buffer** (including the terminating `\0`), or a pointer to the required capacity in the case that **bufferCapacityInput** is insufficient.
- **buffer** is a pointer to an application-allocated buffer that will be filled with the bound source name. It **can** be `NULL` if **bufferCapacityInput** is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required **buffer** size.

[xrGetInputSourceLocalizedName](#) returns a string for the bound source in the current system locale.

If [xrAttachSessionActionSets](#) has not yet been called for the session, the runtime **must** return `XR_ERROR_ACTIONSET_NOT_ATTACHED`.

## Valid Usage (Implicit)

- `session` **must** be a valid `XrSession` handle
- `getInfo` **must** be a pointer to a valid `XrInputSourceLocalizedNameGetInfo` structure
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The `XrInputSourceLocalizedNameGetInfo` structure is defined as:

```
typedef struct XrInputSourceLocalizedNameGetInfo {
    XrStructureType          type;
    const void*             next;
    XrPath                  sourcePath;
    XrInputSourceLocalizedNameFlags whichComponents;
} XrInputSourceLocalizedNameGetInfo;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `sourcePath` is an [XrPath](#) representing a bound source returned by [xrEnumerateBoundSourcesForAction](#).
- `whichComponents` is any set of flags from [XrInputSourceLocalizedNameFlagBits](#).

The result of passing an [XrPath](#) `sourcePath` **not** retrieved from [xrEnumerateBoundSourcesForAction](#) is not specified.

## Valid Usage (Implicit)

- `type` **must** be `XR_TYPE_INPUT_SOURCE_LOCALIZED_NAME_GET_INFO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `whichComponents` **must** be a valid combination of [XrInputSourceLocalizedNameFlagBits](#) values
- `whichComponents` **must** not be `0`

The [XrInputSourceLocalizedNameGetInfo::whichComponents](#) parameter is of the following type, and contains a bitwise-OR of one or more of the bits defined in [XrInputSourceLocalizedNameFlagBits](#).

```
typedef XrFlags64 XrInputSourceLocalizedNameFlags;
```

```
// Flag bits for XrInputSourceLocalizedNameFlags
static const XrInputSourceLocalizedNameFlags XR_INPUT_SOURCE_LOCALIZED_NAME_USER_PATH_BIT
= 0x00000001;
static const XrInputSourceLocalizedNameFlags
XR_INPUT_SOURCE_LOCALIZED_NAME_INTERACTION_PROFILE_BIT = 0x00000002;
static const XrInputSourceLocalizedNameFlags XR_INPUT_SOURCE_LOCALIZED_NAME_COMPONENT_BIT
= 0x00000004;
```

The flag bits have the following meanings:

## Flag Descriptions

- `XR_INPUT_SOURCE_LOCALIZED_NAME_USER_PATH_BIT` indicates that the runtime **must** include the user path portion of the string in the result, if available. E.g. `Left Hand`.
- `XR_INPUT_SOURCE_LOCALIZED_NAME_INTERACTION_PROFILE_BIT` indicates that the runtime **must** include the interaction profile portion of the string in the result, if available. E.g. `Vive Controller`.
- `XR_INPUT_SOURCE_LOCALIZED_NAME_COMPONENT_BIT` indicates that the runtime **must** include the input component portion of the string in the result, if available. E.g. `Trigger`.



# Chapter 12. List of Current Extensions

- `XR_KHR_android_create_instance`
- `XR_KHR_android_surface_swapchain`
- `XR_KHR_android_thread_settings`
- `XR_KHR_binding_modification`
- `XR_KHR_composition_layer_color_scale_bias`
- `XR_KHR_composition_layer_cube`
- `XR_KHR_composition_layer_cylinder`
- `XR_KHR_composition_layer_depth`
- `XR_KHR_composition_layer_equirect`
- `XR_KHR_composition_layer_equirect2`
- `XR_KHR_convert_timespec_time`
- `XR_KHR_D3D11_enable`
- `XR_KHR_D3D12_enable`
- `XR_KHR_loader_init`
- `XR_KHR_loader_init_android`
- `XR_KHR_opengl_enable`
- `XR_KHR_opengl_es_enable`
- `XR_KHR_swapchain_usage_input_attachment_bit`
- `XR_KHR_visibility_mask`
- `XR_KHR_vulkan_enable`
- `XR_KHR_vulkan_enable2`
- `XR_KHR_vulkan_swapchain_format_list`
- `XR_KHR_win32_convert_performance_counter_time`
- `XR_EXT_active_action_set_priority`
- `XR_EXT_conformance_automation`
- `XR_EXT_debug_utils`
- `XR_EXT_dpad_binding`
- `XR_EXT_eye_gaze_interaction`
- `XR_EXT_future`
- `XR_EXT_hand_interaction`

- XR\_EXT\_hand\_joints\_motion\_range
- XR\_EXT\_hand\_tracking
- XR\_EXT\_hand\_tracking\_data\_source
- XR\_EXT\_performance\_settings
- XR\_EXT\_plane\_detection
- XR\_EXT\_thermal\_query
- XR\_EXT\_user\_presence
- XR\_EXT\_view\_configuration\_depth\_range
- XR\_EXT\_win32\_appcontainer\_compatible
- XR\_ALMALENCE\_digital\_lens\_control
- XR\_EPIC\_view\_configuration\_fov
- XR\_FB\_android\_surface\_swapchain\_create
- XR\_FB\_body\_tracking
- XR\_FB\_color\_space
- XR\_FB\_composition\_layer\_alpha\_blend
- XR\_FB\_composition\_layer\_depth\_test
- XR\_FB\_composition\_layer\_image\_layout
- XR\_FB\_composition\_layer\_secure\_content
- XR\_FB\_composition\_layer\_settings
- XR\_FB\_display\_refresh\_rate
- XR\_FB\_eye\_tracking\_social
- XR\_FB\_face\_tracking
- XR\_FB\_face\_tracking2
- XR\_FB\_foveation
- XR\_FB\_foveation\_configuration
- XR\_FB\_foveation\_vulkan
- XR\_FB\_hand\_tracking\_aim
- XR\_FB\_hand\_tracking\_capsules
- XR\_FB\_hand\_tracking\_mesh
- XR\_FB\_haptic\_amplitude\_envelope
- XR\_FB\_haptic\_pcm
- XR\_FB\_keyboard\_tracking

- XR\_FB\_passthrough
- XR\_FB\_passthrough\_keyboard\_hands
- XR\_FB\_render\_model
- XR\_FB\_scene
- XR\_FB\_scene\_capture
- XR\_FB\_space\_warp
- XR\_FB\_spatial\_entity
- XR\_FB\_spatial\_entity\_container
- XR\_FB\_spatial\_entity\_query
- XR\_FB\_spatial\_entity\_sharing
- XR\_FB\_spatial\_entity\_storage
- XR\_FB\_spatial\_entity\_storage\_batch
- XR\_FB\_spatial\_entity\_user
- XR\_FB\_swapchain\_update\_state
- XR\_FB\_swapchain\_update\_state\_android\_surface
- XR\_FB\_swapchain\_update\_state\_opengl\_es
- XR\_FB\_swapchain\_update\_state\_vulkan
- XR\_FB\_touch\_controller\_pro
- XR\_FB\_touch\_controller\_proximity
- XR\_FB\_triangle\_mesh
- XR\_HTC\_anchor
- XR\_HTC\_facial\_tracking
- XR\_HTC\_foveation
- XR\_HTC\_hand\_interaction
- XR\_HTC\_passthrough
- XR\_HTC\_vive\_wrist\_tracker\_interaction
- XR\_HUAWEI\_controller\_interaction
- XR\_META\_automatic\_layer\_filter
- XR\_META\_environment\_depth
- XR\_META\_foveation\_eye\_tracked
- XR\_META\_headset\_id
- XR\_META\_local\_dimming

- XR\_META\_passthrough\_color\_lut
- XR\_META\_passthrough\_preferences
- XR\_META\_performance\_metrics
- XR\_META\_recommended\_layer\_resolution
- XR\_META\_spatial\_entity\_mesh
- XR\_META\_touch\_controller\_plus
- XR\_META\_virtual\_keyboard
- XR\_META\_vulkan\_swapchain\_create\_info
- XR\_ML\_compat
- XR\_ML\_frame\_end\_info
- XR\_ML\_global\_dimmer
- XR\_ML\_localization\_map
- XR\_ML\_marker\_understanding
- XR\_ML\_user\_calibration
- XR\_MND\_headless
- XR\_MSFT\_composition\_layer\_reprojection
- XR\_MSFT\_controller\_model
- XR\_MSFT\_first\_person\_observer
- XR\_MSFT\_hand\_interaction
- XR\_MSFT\_hand\_tracking\_mesh
- XR\_MSFT\_holographic\_window\_attachment
- XR\_MSFT\_perception\_anchor\_interop
- XR\_MSFT\_scene\_marker
- XR\_MSFT\_scene\_understanding
- XR\_MSFT\_scene\_understanding\_serialization
- XR\_MSFT\_secondary\_view\_configuration
- XR\_MSFT\_spatial\_anchor
- XR\_MSFT\_spatial\_anchor\_persistence
- XR\_MSFT\_spatial\_graph\_bridge
- XR\_MSFT\_unbounded\_reference\_space
- XR\_OCULUS\_audio\_device\_guid
- XR\_OCULUS\_external\_camera

- `XR_OPPO_controller_interaction`
- `XR_QCOM_tracking_optimization_settings`
- `XR_ULTRALEAP_hand_tracking_forearm`
- `XR_VALVE_analog_threshold`
- `XR_VARJO_composition_layer_depth_test`
- `XR_VARJO_environment_depth_estimation`
- `XR_VARJO_foveated_rendering`
- `XR_VARJO_marker_tracking`
- `XR_VARJO_view_offset`
- `XR_VARJO_xr4_controller_interaction`
- `XR_YVR_controller_interaction`

# 12.1. XR\_KHR\_android\_create\_instance

## Name String

`XR_KHR_android_create_instance`

## Extension Type

Instance extension

## Registered Extension Number

9

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2019-07-17

## IP Status

No known IP claims.

## Contributors

Robert Menzel, NVIDIA  
Martin Renschler, Qualcomm  
Krzysztof Kosiński, Google

## Overview

When the application creates an [XrInstance](#) object on Android systems, additional information from the application has to be provided to the XR runtime.

The Android XR runtime **must** return error `XR_ERROR_VALIDATION_FAILURE` if the additional information is not provided by the application or if the additional parameters are invalid.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_INSTANCE_CREATE_INFO_ANDROID_KHR`

## New Enums

## New Structures

The [XrInstanceCreateInfoAndroidKHR](#) structure is defined as:

```
// Provided by XR_KHR_android_create_instance
typedef struct XrInstanceCreateInfoAndroidKHR {
    XrStructureType    type;
    const void*        next;
    void*              applicationVM;
    void*              applicationActivity;
} XrInstanceCreateInfoAndroidKHR;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `applicationVM` is a pointer to the JNI's opaque [JavaVM](#) structure, cast to a void pointer.
- `applicationActivity` is a JNI reference to an [android.app.Activity](#) that will drive the session lifecycle of this instance, cast to a void pointer.

[XrInstanceCreateInfoAndroidKHR](#) contains additional Android specific information needed when calling [xrCreateInstance](#). The `applicationVM` field should be populated with the [JavaVM](#) structure received by the `JNI_OnLoad` function, while the `applicationActivity` field will typically contain a reference to a Java activity object received through an application-specific native method. The [XrInstanceCreateInfoAndroidKHR](#) structure **must** be provided in the `next` chain of the [XrInstanceCreateInfo](#) structure when calling [xrCreateInstance](#).

### Valid Usage (Implicit)

- The [XR\\_KHR\\_android\\_create\\_instance](#) extension **must** be enabled prior to using [XrInstanceCreateInfoAndroidKHR](#)
- `type` **must** be `XR_TYPE_INSTANCE_CREATE_INFO_ANDROID_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `applicationVM` **must** be a pointer value
- `applicationActivity` **must** be a pointer value

## New Functions

## Issues

## Version History

- Revision 1, 2017-05-26 (Robert Menzel)
  - Initial draft
- Revision 2, 2019-01-24 (Martin Renschler)
  - Added error code, reformatted
- Revision 3, 2019-07-17 (Krzysztof Kosiński)
  - Non-substantive clarifications.

## 12.2. XR\_KHR\_android\_surface\_swapchain

### Name String

`XR_KHR_android_surface_swapchain`

### Extension Type

Instance extension

### Registered Extension Number

5

### Revision

4

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-05-30

### IP Status

No known IP claims.

### Contributors

Krzysztof Kosiński, Google  
Johannes van Waveren, Oculus  
Martin Renschler, Qualcomm

### Overview

A common activity in XR is to view an image stream. Image streams are often the result of camera



previews or decoded video streams. On Android, the basic primitive representing the producer end of an image queue is the class `android.view.Surface`. This extension provides a special swapchain that uses an `android.view.Surface` as its producer end.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

To create an `XrSwapchain` object and an Android Surface object call:

```
// Provided by XR_KHR_android_surface_swapchain
XrResult xrCreateSwapchainAndroidSurfaceKHR(
    XrSession          session,
    const XrSwapchainCreateInfo* info,
    XrSwapchain*      swapchain,
    jobject*          surface);
```

### Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `info` is a pointer to an `XrSwapchainCreateInfo` structure.
- `swapchain` is a pointer to a handle in which the created `XrSwapchain` is returned.
- `surface` is a pointer to a `jobject` where the created Android Surface is returned.

`xrCreateSwapchainAndroidSurfaceKHR` creates an `XrSwapchain` object returned in `swapchain` and an Android Surface `jobject` returned in `surface`. The `jobject` **must** be valid to be passed back to Java code using JNI and **must** be valid to be used with ordinary Android APIs for submitting images to Surfaces. The returned `XrSwapchain` **must** be valid to be referenced in `XrSwapchainSubImage` structures to show content on the screen. The width and height passed in `XrSwapchainCreateInfo` **may** not be persistent throughout the life cycle of the created swapchain, since on Android, the size of the images is controlled by the producer and possibly changes at any time.

The only function that is allowed to be called on the `XrSwapchain` returned from this function is `xrDestroySwapchain`. For example, calling any of the functions `xrEnumerateSwapchainImages`,

`xrAcquireSwapchainImage`, `xrWaitSwapchainImage` or `xrReleaseSwapchainImage` is invalid.

When the application receives the `XrEventDataSessionStateChanged` event with the `XR_SESSION_STATE_STOPPING` state, it **must** ensure that no threads are writing to any of the Android surfaces created with this extension before calling `xrEndSession`. The effect of writing frames to the Surface when the session is in states other than `XR_SESSION_STATE_VISIBLE` or `XR_SESSION_STATE_FOCUSED` is undefined.

`xrCreateSwapchainAndroidSurfaceKHR` **must** return the same set of error codes as `xrCreateSwapchain` under the same circumstances, plus `XR_ERROR_FUNCTION_UNSUPPORTED` in case the function is not supported.

### Valid Usage of `XrSwapchainCreateInfo` members

- The `XrSwapchainCreateInfo::format`, `XrSwapchainCreateInfo::sampleCount`, `XrSwapchainCreateInfo::faceCount`, `XrSwapchainCreateInfo::arraySize` and `XrSwapchainCreateInfo::mipCount` members of the structure passed as the `info` parameter **must** be zero.

### Valid Usage (Implicit)

- The `XR_KHR_android_surface_swapchain` extension **must** be enabled prior to calling `xrCreateSwapchainAndroidSurfaceKHR`
- `session` **must** be a valid `XrSession` handle
- `info` **must** be a pointer to a valid `XrSwapchainCreateInfo` structure
- `swapchain` **must** be a pointer to an `XrSwapchain` handle
- `surface` **must** be a pointer to a `jobject` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

## Issues

### Version History

- Revision 1, 2017-01-17 (Johannes van Waveren)
  - Initial draft
- Revision 2, 2017-10-30 (Kaye Mason)
  - Changed images to swapchains, used snippet includes. Added issue for Surfaces.
- Revision 3, 2018-05-16 (Krzysztof Kosiński)
  - Refactored to use Surface instead of SurfaceTexture.
- Revision 4, 2019-01-24 (Martin Renschler)
  - Refined the specification of the extension

## 12.3. XR\_KHR\_android\_thread\_settings

### Name String

`XR_KHR_android_thread_settings`

### Extension Type

Instance extension

## Registered Extension Number

4

## Revision

6

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2023-12-04

## IP Status

No known IP claims.

## Contributors

Cass Everitt, Oculus

Johannes van Waveren, Oculus

Martin Renschler, Qualcomm

Krzysztof Kosiński, Google

Xiang Wei, Meta

## Overview

For XR to be comfortable, it is important for applications to deliver frames quickly and consistently. In order to make sure the important application threads get their full share of time, these threads must be identified to the system, which will adjust their scheduling priority accordingly.

## New Object Types

## New Flag Types

## New Enum Constants

[XrResult](#) enumeration is extended with:

- `XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR`
- `XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR`

## New Enums

The possible thread types are specified by the [XrAndroidThreadTypeKHR](#) enumeration:

```
// Provided by XR_KHR_android_thread_settings
typedef enum XrAndroidThreadTypeKHR {
    XR_ANDROID_THREAD_TYPE_APPLICATION_MAIN_KHR = 1,
    XR_ANDROID_THREAD_TYPE_APPLICATION_WORKER_KHR = 2,
    XR_ANDROID_THREAD_TYPE_RENDERER_MAIN_KHR = 3,
    XR_ANDROID_THREAD_TYPE_RENDERER_WORKER_KHR = 4,
    XR_ANDROID_THREAD_TYPE_MAX_ENUM_KHR = 0x7FFFFFFF
} XrAndroidThreadTypeKHR;
```

## Enumerants

- **XR\_ANDROID\_THREAD\_TYPE\_APPLICATION\_MAIN\_KHR**  
hints the XR runtime that the thread is doing time critical CPU tasks
- **XR\_ANDROID\_THREAD\_TYPE\_APPLICATION\_WORKER\_KHR**  
hints the XR runtime that the thread is doing background CPU tasks
- **XR\_ANDROID\_THREAD\_TYPE\_RENDERER\_MAIN\_KHR**  
hints the XR runtime that the thread is doing time critical graphics device tasks
- **XR\_ANDROID\_THREAD\_TYPE\_RENDERER\_WORKER\_KHR**  
hints the XR runtime that the thread is doing background graphics device tasks

## New Structures

## New Functions

To declare a thread to be of a certain [XrAndroidThreadTypeKHR](#) type call:

```
// Provided by XR_KHR_android_thread_settings
XrResult xrSetAndroidApplicationThreadKHR(
    XrSession session,
    XrAndroidThreadTypeKHR threadType,
    uint32_t threadId);
```

## Parameter Descriptions

- `session` is a valid [XrSession](#) handle.
- `threadType` is a classification of the declared thread allowing the XR runtime to apply the relevant priority and attributes. If such settings fail, the runtime **must** return `XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR`.
- `threadId` is the kernel thread ID of the declared thread, as returned by `gettid()` or `android.os.process.myTid()`. If the thread ID is invalid, the runtime **must** return `XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR`.

[xrSetAndroidApplicationThreadKHR](#) allows to declare an XR-critical thread and to classify it.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_android\\_thread\\_settings](#) extension **must** be enabled prior to calling [xrSetAndroidApplicationThreadKHR](#)
- `session` **must** be a valid [XrSession](#) handle
- `threadType` **must** be a valid [XrAndroidThreadTypeKHR](#) value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR`
- `XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR`

## Version History

- Revision 1, 2017-01-17 (Johannes van Waveren)
  - Initial draft.
- Revision 2, 2017-10-31 (Armelle Laine)
  - Move the performance settings to EXT extension.
- Revision 3, 2018-12-20 (Paul Pedriana)
  - Revised the error code naming to use KHR and renamed `xrSetApplicationThreadKHR` → [xrSetAndroidApplicationThreadKHR](#).
- Revision 4, 2019-01-24 (Martin Renschler)
  - Added enum specification, reformatting
- Revision 5, 2019-07-17 (Krzysztof Kosiński)
  - Clarify the type of thread identifier used by the extension.
- Revision 6, 2023-12-04 (Xiang Wei)
  - Revise/fix the hints of enum specification

## 12.4. XR\_KHR\_binding\_modification

### Name String

`XR_KHR_binding_modification`

### Extension Type

Instance extension

### Registered Extension Number

121

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2020-07-29

### IP Status

No known IP claims.

### Contributors

Joe Ludwig, Valve

## Contacts

Joe Ludwig, Valve

## Overview

This extension adds an optional structure that can be included on the [XrInteractionProfileSuggestedBinding::next](#) chain passed to [xrSuggestInteractionProfileBindings](#) to specify additional information to modify default binding behavior.

This extension does not define any actual modification structs, but includes the list of modifications and the [XrBindingModificationBaseHeaderKHR](#) structure to allow other extensions to provide specific modifications.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_BINDING\\_MODIFICATIONS\\_KHR](#)

## New Enums

## New Structures

The [XrBindingModificationsKHR](#) structure is defined as:

```
// Provided by XR_KHR_binding_modification
typedef struct XrBindingModificationsKHR {
    XrStructureType                type;
    const void*                    next;
    uint32_t                       bindingModificationCount;
    const XrBindingModificationBaseHeaderKHR* const* bindingModifications;
} XrBindingModificationsKHR;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `bindingModificationCount` is the number of binding modifications in the array pointed to by `bindingModifications`.
- `bindingModifications` is a pointer to an array of pointers to binding modification structures based on [XrBindingModificationBaseHeaderKHR](#), that define all of the application's suggested binding modifications for the specified interaction profile.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_binding\\_modification](#) extension **must** be enabled prior to using [XrBindingModificationsKHR](#)
- `type` **must** be `XR_TYPE_BINDING_MODIFICATIONS_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `bindingModificationCount` is not `0`, `bindingModifications` **must** be a pointer to an array of `bindingModificationCount` valid [XrBindingModificationBaseHeaderKHR](#)-based structures. See also: [XrInteractionProfileAnalogThresholdVALVE](#), [XrInteractionProfileDpadBindingEXT](#)

The [XrBindingModificationBaseHeaderKHR](#) structure is defined as:

```
// Provided by XR_KHR_binding_modification
typedef struct XrBindingModificationBaseHeaderKHR {
    XrStructureType    type;
    const void*        next;
} XrBindingModificationBaseHeaderKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or in this extension.

The [XrBindingModificationBaseHeaderKHR](#) is a base structure is overridden by `XrBindingModification*`

child structures.

### Valid Usage (Implicit)

- The [XR\\_KHR\\_binding\\_modification](#) extension **must** be enabled prior to using [XrBindingModificationBaseHeaderKHR](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### New Functions

### Issues

### Version History

- Revision 1, 2020-08-06 (Joe Ludwig)
  - Initial draft.

## 12.5. XR\_KHR\_composition\_layer\_color\_scale\_bias

### Name String

`XR_KHR_composition_layer_color_scale_bias`

### Extension Type

Instance extension

### Registered Extension Number

35

### Revision

5

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-01-28

### IP Status

No known IP claims.

### Contributors

Paul Pedriana, Oculus  
Cass Everitt, Oculus

## Overview

Color scale and bias are applied to a layer color during composition, after its conversion to premultiplied alpha representation.

If specified, `colorScale` and `colorBias` **must** be used to alter the `LayerColor` as follows:

- `colorScale = max( vec4( 0, 0, 0, 0 ), colorScale )`
- `LayerColor.RGB = LayerColor.A > 0 ? LayerColor.RGB / LayerColor.A : vec3( 0, 0, 0 )`
- `LayerColor = LayerColor * colorScale + colorBias`
- `LayerColor.RGB *= LayerColor.A`

This extension specifies the `XrCompositionLayerColorScaleBiasKHR` structure, which, if present in the `XrCompositionLayerBaseHeader::next` chain, **must** be applied to the composition layer.

This extension does not define a new composition layer type, but rather it defines a transform that may be applied to the color derived from existing composition layer types.

## New Object Types

## New Flag Types

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_COLOR_SCALE_BIAS_KHR`

## New Enums

## New Structures

The `XrCompositionLayerColorScaleBiasKHR` structure is defined as:

```
// Provided by XR_KHR_composition_layer_color_scale_bias
typedef struct XrCompositionLayerColorScaleBiasKHR {
    XrStructureType    type;
    const void*        next;
    XrColor4f          colorScale;
    XrColor4f          colorBias;
} XrCompositionLayerColorScaleBiasKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `colorScale` is an [XrColor4f](#) which will modulate the color sourced from the images.
- `colorBias` is an [XrColor4f](#) which will offset the color sourced from the images.

[XrCompositionLayerColorScaleBiasKHR](#) contains the information needed to scale and bias the color of layer textures.

The [XrCompositionLayerColorScaleBiasKHR](#) structure **can** be applied by applications to composition layers by adding an instance of the struct to the [XrCompositionLayerBaseHeader::next](#) list.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_composition\\_layer\\_color\\_scale\\_bias](#) extension **must** be enabled prior to using [XrCompositionLayerColorScaleBiasKHR](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_COLOR_SCALE_BIAS_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

## Issues

## Version History

- Revision 1, 2017-09-13 (Paul Pedriana)
  - Initial implementation.
- Revision 2, 2019-01-24 (Martin Renschler)
  - Formatting, spec language changes
- Revision 3, 2019-01-28 (Paul Pedriana)
  - Revised math to remove pre-multiplied alpha before applying color scale and offset, then restoring.
- Revision 4, 2019-07-17 (Cass Everitt)
  - Non-substantive updates to the spec language and equations.
- Revision 5, 2020-05-20 (Cass Everitt)
  - Changed extension name, simplified language.

# 12.6. XR\_KHR\_composition\_layer\_cube

## Name String

XR\_KHR\_composition\_layer\_cube

## Extension Type

Instance extension

## Registered Extension Number

7

## Revision

8

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2019-01-24

## IP Status

No known IP claims.

## Contributors

Johannes van Waveren, Oculus  
Cass Everitt, Oculus  
Paul Pedriana, Oculus  
Gloria Kennickell, Oculus  
Sam Martin, ARM  
Kaye Mason, Google, Inc.  
Martin Renschler, Qualcomm

## Contacts

Cass Everitt, Oculus  
Paul Pedriana, Oculus

## Overview

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.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_CUBE_KHR`

## New Enums

## New Structures

The [XrCompositionLayerCubeKHR](#) structure is defined as:

```
// Provided by XR_KHR_composition_layer_cube
typedef struct XrCompositionLayerCubeKHR {
    XrStructureType      type;
    const void*          next;
    XrCompositionLayerFlags layerFlags;
    XrSpace              space;
    XrEyeVisibility      eyeVisibility;
    XrSwapchain          swapchain;
    uint32_t             imageArrayIndex;
    XrQuaternionf        orientation;
} XrCompositionLayerCubeKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layerFlags` is any flags to apply to this layer.
- `space` is the [XrSpace](#) in which the `orientation` of the cube layer is evaluated over time.
- `eyeVisibility` is the eye represented by this layer.
- `swapchain` is the swapchain, which **must** have been created with a [XrSwapchainCreateInfo](#) `::faceCount` of 6.
- `imageArrayIndex` is the image array index, with 0 meaning the first or only array element.
- `orientation` is the orientation of the environment map in the `space`.

[XrCompositionLayerCubeKHR](#) contains the information needed to render a cube map when calling

`xrEndFrame`. `XrCompositionLayerCubeKHR` is an alias type for the base struct `XrCompositionLayerBaseHeader` used in `XrFrameEndInfo`.

### Valid Usage (Implicit)

- The `XR_KHR_composition_layer_cube` extension **must** be enabled prior to using `XrCompositionLayerCubeKHR`
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_CUBE_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layerFlags` **must** be `0` or a valid combination of `XrCompositionLayerFlagBits` values
- `space` **must** be a valid `XrSpace` handle
- `eyeVisibility` **must** be a valid `XrEyeVisibility` value
- `swapchain` **must** be a valid `XrSwapchain` handle
- Both of `space` and `swapchain` **must** have been created, allocated, or retrieved from the same `XrSession`

### New Functions

### Issues

### Version History

- Revision 0, 2017-02-01 (Johannes van Waveren)
  - Initial draft.
- Revision 1, 2017-05-19 (Sam Martin)
  - Initial draft, moving the 3 layer types to an extension.
- Revision 2, 2017-08-30 (Paul Pedriana)
  - Updated the specification.
- Revision 3, 2017-10-12 (Cass Everitt)
  - Updated to reflect per-eye structs and the change to swapchains
- Revision 4, 2017-10-18 (Kaye Mason)
  - Update to flatten structs to remove per-eye arrays.
- Revision 5, 2017-12-05 (Paul Pedriana)
  - Updated to break out the cylinder and equirect features into separate extensions.
- Revision 6, 2017-12-07 (Paul Pedriana)
  - Updated to use transform components instead of transform matrices.

- Revision 7, 2017-12-07 (Paul Pedriana)
  - Updated to convert [XrPosef](#) to [XrQuaternionf](#) (there's no position component).
- Revision 8, 2019-01-24 (Martin Renschler)
  - Updated struct to use [XrSwapchainSubImage](#), reformat and spec language changes, eye parameter description update

## 12.7. XR\_KHR\_composition\_layer\_cylinder

### Name String

`XR_KHR_composition_layer_cylinder`

### Extension Type

Instance extension

### Registered Extension Number

18

### Revision

4

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-01-24

### IP Status

No known IP claims.

### Contributors

James Hughes, Oculus

Paul Pedriana, Oculus

Martin Renschler, Qualcomm

### Contacts

Paul Pedriana, Oculus

Cass Everitt, Oculus

### Overview

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.

The cylinder characteristics are specified by the following parameters:

```
XrPosef      pose;  
float        radius;  
float        centralAngle;  
float        aspectRatio;
```

These can be understood via the following diagram, which is a top-down view of a horizontally oriented cylinder. The aspect ratio drives how tall the cylinder will appear based on the other parameters. Typically the aspectRatio would be set to be the aspect ratio of the texture being used, so that it looks the same within the cylinder as it does in 2D.

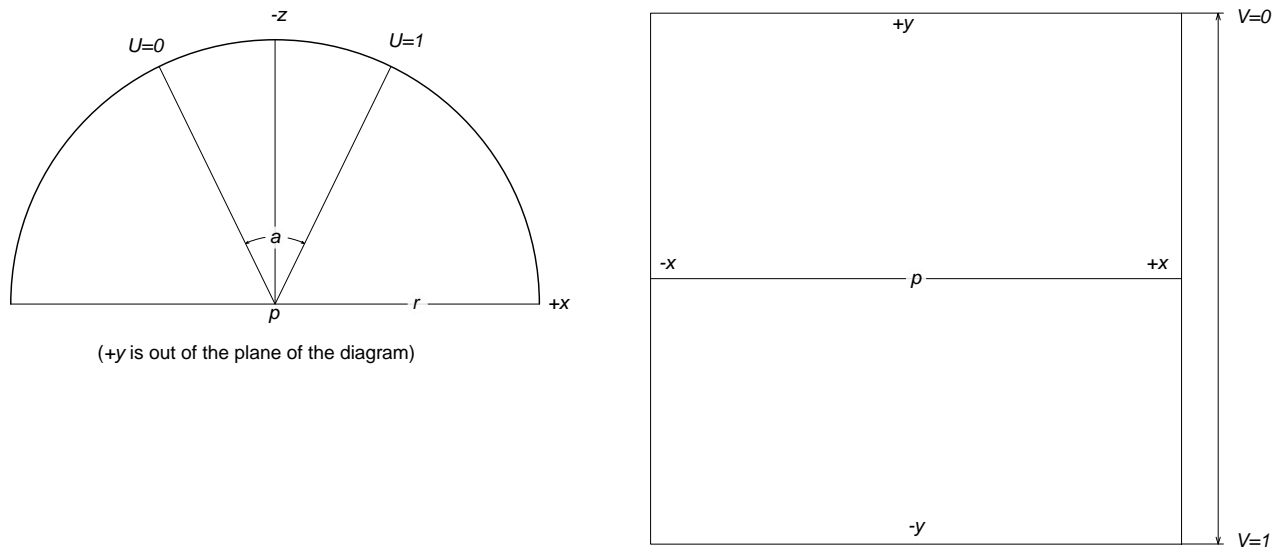


Figure 6. Cylinder Layer Parameters

- $r$  — Radius
- $a$  — Central angle in  $(0, 2\pi)$
- $p$  — Origin of pose transform
- $U/V$  — UV coordinates

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_CYLINDER_KHR`

## New Enums

## New Structures

The `XrCompositionLayerCylinderKHR` structure is defined as:

```
// Provided by XR_KHR_composition_layer_cylinder
typedef struct XrCompositionLayerCylinderKHR {
    XrStructureType      type;
    const void*         next;
    XrCompositionLayerFlags layerFlags;
    XrSpace              space;
    XrEyeVisibility     eyeVisibility;
    XrSwapchainSubImage subImage;
    XrPosef             pose;
    float               radius;
    float               centralAngle;
    float               aspectRatio;
} XrCompositionLayerCylinderKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layerFlags` specifies options for the layer.
- `space` is the [XrSpace](#) in which the `pose` of the cylinder layer is evaluated over time.
- `eyeVisibility` is the eye represented by this layer.
- `subImage` identifies the image [XrSwapchainSubImage](#) to use. The swapchain **must** have been created with a [XrSwapchainCreateInfo::faceCount](#) of 1.
- `pose` is an [XrPosef](#) defining the position and orientation of the center point of the view of the cylinder within the reference frame of the `space`.
- `radius` is the non-negative radius of the cylinder. Values of zero or floating point positive infinity are treated as an infinite cylinder.
- `centralAngle` is the angle of the visible section of the cylinder, based at 0 radians, in the range of  $[0, 2\pi)$ . It grows symmetrically around the 0 radian angle.
- `aspectRatio` is the ratio of the visible cylinder section width / height. The height of the cylinder is given by:  $(\text{cylinder radius} \times \text{cylinder angle}) / \text{aspectRatio}$ .

[XrCompositionLayerCylinderKHR](#) contains the information needed to render a texture onto a cylinder when calling `xrEndFrame`. [XrCompositionLayerCylinderKHR](#) is an alias type for the base struct [XrCompositionLayerBaseHeader](#) used in [XrFrameEndInfo](#).

## Valid Usage (Implicit)

- The `XR_KHR_composition_layer_cylinder` extension **must** be enabled prior to using [XrCompositionLayerCylinderKHR](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_CYLINDER_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layerFlags` **must** be `0` or a valid combination of [XrCompositionLayerFlagBits](#) values
- `space` **must** be a valid [XrSpace](#) handle
- `eyeVisibility` **must** be a valid [XrEyeVisibility](#) value
- `subImage` **must** be a valid [XrSwapchainSubImage](#) structure

## New Functions

## Issues

## Version History

- Revision 1, 2017-05-19 (Paul Pedriana)
  - Initial version. This was originally part of a single extension which supported multiple such extension layer types.
- Revision 2, 2017-12-07 (Paul Pedriana)
  - Updated to use transform components instead of transform matrices.
- Revision 3, 2018-03-05 (Paul Pedriana)
  - Added improved documentation and brought the documentation in line with the existing core spec.
- Revision 4, 2019-01-24 (Martin Renschler)
  - Reformatted, spec language changes, eye parameter description update

## 12.8. XR\_KHR\_composition\_layer\_depth

### Name String

`XR_KHR_composition_layer_depth`

### Extension Type

Instance extension

### Registered Extension Number

11

### Revision

6

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-01-24

### IP Status

No known IP claims.

### Contributors

Paul Pedriana, Oculus  
Bryce Hutchings, Microsoft  
Andreas Loeve Selvik, Arm  
Martin Renschler, Qualcomm

## Overview

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](#).

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_DEPTH_INFO_KHR`

## New Enums

## New Structures

When submitting depth images along with projection layers, add the [XrCompositionLayerDepthInfoKHR](#) to the `next` chain for all [XrCompositionLayerProjectionView](#) structures in the given layer.

The [XrCompositionLayerDepthInfoKHR](#) structure is defined as:

```
// Provided by XR_KHR_composition_layer_depth
typedef struct XrCompositionLayerDepthInfoKHR {
    XrStructureType    type;
    const void*        next;
    XrSwapchainSubImage subImage;
    float               minDepth;
    float               maxDepth;
    float               nearZ;
    float               farZ;
} XrCompositionLayerDepthInfoKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `subImage` identifies the depth image [XrSwapchainSubImage](#) to be associated with the color swapchain. The swapchain **must** have been created with a [XrSwapchainCreateInfo::faceCount](#) of 1.
- `minDepth` and `maxDepth` are the window space depths that correspond to the near and far frustum planes, respectively. `minDepth` must be less than `maxDepth`. `minDepth` and `maxDepth` must be in the range [0, 1].
- `nearZ` and `farZ` are the positive distances in meters to the near and far frustum planes, respectively. `nearZ` and `farZ` **must** not be equal. `nearZ` and `farZ` **must** be in the range (0, +infinity].

### Note



The window space depth values `minDepth` and `maxDepth` are akin to the parameters of [glDepthRange](#) that specify the mapping from normalized device coordinates into window space.

### Note



A reversed mapping of depth, such that points closer to the view have a window space depth that is greater than points further away can be achieved by making `nearZ > farZ`.

[XrCompositionLayerDepthInfoKHR](#) contains the information needed to associate depth with the color information in a projection layer. When submitting depth images along with projection layers, add the [XrCompositionLayerDepthInfoKHR](#) to the `next` chain for all [XrCompositionLayerProjectionView](#) structures in the given layer.

The homogeneous transform from view space  $z$  to window space depth is given by the following matrix, where  $a = \text{minDepth}$ ,  $b = \text{maxDepth}$ ,  $n = \text{nearZ}$ , and  $f = \text{farZ}$ .

$$\mathbf{T} = \begin{bmatrix} b - a & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{f}{f-n} & -\frac{fn}{f-n} \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -\frac{bf-an}{f-n} & -\frac{fn(b-a)}{f-n} \\ -1 & 0 \end{bmatrix}$$

$$\mathbf{p}_w = \mathbf{T}\mathbf{p}_v$$

$$\mathbf{p}_w = [z_w \quad w_w]^t, \text{ homogeneous window space depth}$$

$$\mathbf{p}_v = [z_v \quad w_v]^t, \text{ homogeneous view space depth}$$

Figure 7. Homogeneous transform from view space to window space depth

Homogeneous values are constructed from real values by appending a w component with value 1.0.

General homogeneous values are projected back to real space by dividing by the w component.

### Valid Usage (Implicit)

- The [XR\\_KHR\\_composition\\_layer\\_depth](#) extension **must** be enabled prior to using [XrCompositionLayerDepthInfoKHR](#)
- **type** **must** be [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_DEPTH\\_INFO\\_KHR](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)
- **subImage** **must** be a valid [XrSwapchainSubImage](#) structure

### New Functions

### Issues

1. Should the range of [minDepth](#) and [maxDepth](#) be constrained to [0,1]?

**RESOLVED:** Yes.

There is no compelling mathematical reason for this constraint, however, it does not impose any hardship currently, and the constraint could be relaxed in a future version of the extension if needed.

2. Should we require [minDepth](#) be less than [maxDepth](#)?

**RESOLVED:** Yes.

There is no compelling mathematical reason for this constraint, however, it does not impose any

hardship currently, and the constraint could be relaxed in a future version of the extension if needed. Reverse z mappings can be achieved by making `nearZ > farZ`.

3. Does this extension support view space depth images?

**RESOLVED:** No.

The formulation of the transform between view and window depths implies projected depth. A different extension would be needed to support a different interpretation of depth.

4. Is there any constraint on the resolution of the depth subimage?

**RESOLVED:** No.

The resolution of the depth image need not match that of the corresponding color image.

## Version History

- Revision 1, 2017-08-18 (Paul Pedriana)
  - Initial proposal.
- Revision 2, 2017-10-30 (Kaye Mason)
  - Migration from Images to Swapchains.
- Revision 3, 2018-07-20 (Bryce Hutchings)
  - Support for swapchain texture arrays
- Revision 4, 2018-12-17 (Andreas Loeve Selvik)
  - `depthImageRect` in pixels instead of UVs
- Revision 5, 2019-01-24 (Martin Renschler)
  - changed `depthSwapchain/depthImageRect/depthImageArrayIndex` to [XrSwapchainSubImage](#)
  - reformat and spec language changes
  - removed vendor specific terminology
- Revision 6, 2022-02-16 (Cass Everitt)
  - Provide homogeneous transform as function of provided parameters

## 12.9. XR\_KHR\_composition\_layer\_equirect

### Name String

`XR_KHR_composition_layer_equirect`

### Extension Type

Instance extension



## Registered Extension Number

19

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2019-01-24

## IP Status

No known IP claims.

## Contributors

Johannes van Waveren, Oculus  
Cass Everitt, Oculus  
Paul Pedriana, Oculus  
Gloria Kennickell, Oculus  
Martin Renschler, Qualcomm

## Contacts

Cass Everitt, Oculus  
Paul Pedriana, Oculus

## Overview

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.

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 sphere surface **must** be visible; the exterior of the sphere is not visible and **must** not be drawn by the runtime.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_EQUIRECT\\_KHR](#)

## New Enums

## New Structures

The [XrCompositionLayerEquirectKHR](#) structure is defined as:

```
// Provided by XR_KHR_composition_layer_equirect
typedef struct XrCompositionLayerEquirectKHR {
    XrStructureType      type;
    const void*          next;
    XrCompositionLayerFlags layerFlags;
    XrSpace              space;
    XrEyeVisibility      eyeVisibility;
    XrSwapchainSubImage subImage;
    XrPosef              pose;
    float                radius;
    XrVector2f           scale;
    XrVector2f           bias;
} XrCompositionLayerEquirectKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layerFlags` specifies options for the layer.
- `space` is the [XrSpace](#) in which the `pose` of the equirect layer is evaluated over time.
- `eyeVisibility` is the eye represented by this layer.
- `subImage` identifies the image [XrSwapchainSubImage](#) to use. The swapchain **must** have been created with a [XrSwapchainCreateInfo::faceCount](#) of 1.
- `pose` is an [XrPosef](#) defining the position and orientation of the center point of the sphere onto which the equirect image data is mapped, relative to the reference frame of the `space`.
- `radius` is the non-negative radius of the sphere onto which the equirect image data is mapped. Values of zero or floating point positive infinity are treated as an infinite sphere.
- `scale` is an [XrVector2f](#) indicating a scale of the texture coordinates after the mapping to 2D.
- `bias` is an [XrVector2f](#) indicating a bias of the texture coordinates after the mapping to 2D.

[XrCompositionLayerEquirectKHR](#) contains the information needed to render an equirectangular image onto a sphere when calling `xrEndFrame`. [XrCompositionLayerEquirectKHR](#) is an alias type for the base struct [XrCompositionLayerBaseHeader](#) used in [XrFrameEndInfo](#).

## Valid Usage (Implicit)

- The [XR\\_KHR\\_composition\\_layer\\_equirect](#) extension **must** be enabled prior to using [XrCompositionLayerEquirectKHR](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_EQUIRECT_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layerFlags` **must** be `0` or a valid combination of [XrCompositionLayerFlagBits](#) values
- `space` **must** be a valid [XrSpace](#) handle
- `eyeVisibility` **must** be a valid [XrEyeVisibility](#) value
- `subImage` **must** be a valid [XrSwapchainSubImage](#) structure

## New Functions

## Issues

## Version History

- Revision 1, 2017-05-19 (Paul Pedriana)
  - Initial version. This was originally part of a single extension which supported multiple such extension layer types.
- Revision 2, 2017-12-07 (Paul Pedriana)
  - Updated to use transform components instead of transform matrices.
- Revision 3, 2019-01-24 (Martin Renschler)
  - Reformatted, spec language changes, eye parameter description update

## 12.10. XR\_KHR\_composition\_layer\_equirect2

### Name String

`XR_KHR_composition_layer_equirect2`

### Extension Type

Instance extension

### Registered Extension Number

92

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-01-24

### IP Status

No known IP claims.

### Contributors

Johannes van Waveren, Oculus  
Cass Everitt, Oculus  
Paul Pedriana, Oculus  
Gloria Kennickell, Oculus  
Martin Renschler, Qualcomm

### Contacts

Cass Everitt, Oculus

### Overview

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.

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 sphere surface **must** be visible; the exterior of the sphere is not visible and **must** not be drawn by the runtime.

This extension uses a different parameterization more in keeping with the formulation of `KHR_composition_layer_cylinder` but is functionally equivalent to `KHR_composition_layer_equirect`.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_EQUIRECT2_KHR`

## New Enums

## New Structures

The [XrCompositionLayerEquirect2KHR](#) structure is defined as:

```
// Provided by XR_KHR_composition_layer_equirect2
typedef struct XrCompositionLayerEquirect2KHR {
    XrStructureType      type;
    const void*          next;
    XrCompositionLayerFlags layerFlags;
    XrSpace              space;
    XrEyeVisibility      eyeVisibility;
    XrSwapchainSubImage  subImage;
    XrPosef              pose;
    float                radius;
    float                centralHorizontalAngle;
    float                upperVerticalAngle;
    float                lowerVerticalAngle;
} XrCompositionLayerEquirect2KHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layerFlags` specifies options for the layer.
- `space` is the [XrSpace](#) in which the `pose` of the equirect layer is evaluated over time.
- `eyeVisibility` is the eye represented by this layer.
- `subImage` identifies the image [XrSwapchainSubImage](#) to use. The swapchain **must** have been created with a [XrSwapchainCreateInfo::faceCount](#) of 1.
- `pose` is an [XrPosef](#) defining the position and orientation of the center point of the sphere onto which the equirect image data is mapped, relative to the reference frame of the `space`.
- `radius` is the non-negative radius of the sphere onto which the equirect image data is mapped. Values of zero or floating point positive infinity are treated as an infinite sphere.
- `centralHorizontalAngle` defines the visible horizontal angle of the sphere, based at 0 radians, in the range of  $[0, 2\pi]$ . It grows symmetrically around the 0 radian angle.
- `upperVerticalAngle` defines the upper vertical angle of the visible portion of the sphere, in the range of  $[-\pi/2, \pi/2]$ .
- `lowerVerticalAngle` defines the lower vertical angle of the visible portion of the sphere, in the range of  $[-\pi/2, \pi/2]$ .

[XrCompositionLayerEquirect2KHR](#) contains the information needed to render an equirectangular image onto a sphere when calling `xrEndFrame`. [XrCompositionLayerEquirect2KHR](#) is an alias type for the base struct [XrCompositionLayerBaseHeader](#) used in [XrFrameEndInfo](#).

## Valid Usage (Implicit)

- The `XR_KHR_composition_layer_equirect2` extension **must** be enabled prior to using [XrCompositionLayerEquirect2KHR](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_EQUIRECT2_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layerFlags` **must** be `0` or a valid combination of [XrCompositionLayerFlagBits](#) values
- `space` **must** be a valid [XrSpace](#) handle
- `eyeVisibility` **must** be a valid [XrEyeVisibility](#) value
- `subImage` **must** be a valid [XrSwapchainSubImage](#) structure

## New Functions

## Issues

## Version History

- Revision 1, 2020-05-08 (Cass Everitt)
  - Initial version.
  - Kept contributors from the original equirect extension.

# 12.11. XR\_KHR\_convert\_timespec\_time

## Name String

`XR_KHR_convert_timespec_time`

## Extension Type

Instance extension

## Registered Extension Number

37

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2019-01-24

## IP Status

No known IP claims.

## Contributors

Paul Pedriana, Oculus

## Overview

This extension provides two functions for converting between timespec monotonic time and [XrTime](#). The [xrConvertTimespecTimeToTimeKHR](#) function converts from timespec time to [XrTime](#), while the [xrConvertTimeToTimespecTimeKHR](#) function converts [XrTime](#) to timespec monotonic time. The primary use case for this functionality is to be able to synchronize events between the local system and the OpenXR system.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

To convert from timespec monotonic time to [XrTime](#), call:

```
// Provided by XR_KHR_convert_timespec_time
XrResult xrConvertTimespecTimeToTimeKHR(
    XrInstance          instance,
    const struct timespec* timespecTime,
    XrTime*            time);
```

### Parameter Descriptions

- `instance` is an [XrInstance](#) handle previously created with [xrCreateInstance](#).
- `timespecTime` is a `timespec` obtained from `clock_gettime` with `CLOCK_MONOTONIC`.
- `time` is the resulting [XrTime](#) that is equivalent to the `timespecTime`.

The [xrConvertTimespecTimeToTimeKHR](#) function converts a time obtained by the `clock_gettime` function to the equivalent [XrTime](#).

If the output `time` cannot represent the input `timespecTime`, the runtime **must** return `XR_ERROR_TIME_INVALID`.

### Valid Usage (Implicit)

- The `XR_KHR_convert_timespec_time` extension **must** be enabled prior to calling [xrConvertTimespecTimeToTimeKHR](#)
- `instance` **must** be a valid [XrInstance](#) handle
- `timespecTime` **must** be a pointer to a valid `timespec` value
- `time` **must** be a pointer to an [XrTime](#) value



## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_TIME_INVALID`

To convert from `XrTime` to timespec monotonic time, call:

```
// Provided by XR_KHR_convert_timespec_time
XrResult xrConvertTimeToTimespecTimeKHR(
    XrInstance          instance,
    XrTime              time,
    struct timespec*    timespecTime);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `time` is an `XrTime`.
- `timespecTime` is the resulting timespec time that is equivalent to a `timespec` obtained from `clock_gettime` with `CLOCK_MONOTONIC`.

The `xrConvertTimeToTimespecTimeKHR` function converts an `XrTime` to time as if generated by `clock_gettime`.

If the output `timespecTime` cannot represent the input `time`, the runtime **must** return `XR_ERROR_TIME_INVALID`.

## Valid Usage (Implicit)

- The `XR_KHR_convert_timespec_time` extension **must** be enabled prior to calling `xrConvertTimeToTimespecTimeKHR`
- `instance` **must** be a valid `XrInstance` handle
- `timespecTime` **must** be a pointer to a `timespec` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_TIME_INVALID`

## Issues

## Version History

- Revision 1, 2019-01-24 (Paul Pedriana)
  - Initial draft

## 12.12. XR\_KHR\_D3D11\_enable

### Name String

`XR_KHR_D3D11_enable`

### Extension Type

Instance extension

### Registered Extension Number

28

## Revision

9

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2018-11-16

## IP Status

No known IP claims.

## Contributors

Bryce Hutchings, Microsoft

Paul Pedriana, Oculus

Mark Young, LunarG

Minmin Gong, Microsoft

Matthieu Bucchianeri, Microsoft

## Overview

This extension enables the use of the D3D11 graphics API in an OpenXR runtime. Without this extension, the OpenXR runtime may not be able to use any D3D11 swapchain images.

This extension provides the mechanisms necessary for an application to generate a valid [XrGraphicsBindingD3D11KHR](#) structure in order to create a D3D11-based [XrSession](#). Note that during this process the application is responsible for creating all the required D3D11 objects, including a graphics device to be used for rendering.

This extension also provides mechanisms for the application to interact with images acquired by calling [xrEnumerateSwapchainImages](#).

In order to expose the structures, types, and functions of this extension, you **must** define [XR\\_USE\\_GRAPHICS\\_API\\_D3D11](#) before including the OpenXR platform header [openxr\\_platform.h](#), in all portions of your library or application that include it. **Swapchain Flag Bits**

All [XrSwapchainUsageFlags](#) values passed in a session created using [XrGraphicsBindingD3D11KHR](#) **must** be interpreted as follows by the runtime, so that the returned swapchain images used by the application may be used as if they were created with the corresponding D3D11\_BIND\_FLAG flags. The runtime **may** set additional bind flags but **must** not restrict usage.

| <a href="#">XrSwapchainUsageFlagBits</a>                        | Corresponding D3D11 bind flag bits          |
|---|---|
| <a href="#">XR_SWAPCHAIN_USAGE_COLOR_ATTACHMENT_BIT</a>         | <a href="#">D3D11_BIND_RENDER_TARGET</a>    |
| <a href="#">XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT</a> | <a href="#">D3D11_BIND_DEPTH_STENCIL</a>    |
| <a href="#">XR_SWAPCHAIN_USAGE_UNORDERED_ACCESS_BIT</a>         | <a href="#">D3D11_BIND_UNORDERED_ACCESS</a> |

| <b>XrSwapchainUsageFlagBits</b>  | <b>Corresponding D3D11 bind flag bits</b> |
|--|---|
| <code>XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT</code>   | <i>ignored</i>                            |
| <code>XR_SWAPCHAIN_USAGE_TRANSFER_DST_BIT</code>   | <i>ignored</i>                            |
| <code>XR_SWAPCHAIN_USAGE_SAMPLED_BIT</code>  | <code>D3D11_BIND_SHADER_RESOURCE</code>   |
| <code>XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT</code>   | <i>ignored</i>                            |
| <code>XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR</code><br>(Added by <a href="#">XR_KHR_swapchain_usage_input_attachment_bit</a> and only available when that extension is enabled) | <i>ignored</i>                            |

All D3D11 swapchain textures are created with `D3D11_USAGE_DEFAULT` usage.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_GRAPHICS_REQUIREMENTS_D3D11_KHR`
- `XR_TYPE_GRAPHICS_BINDING_D3D11_KHR`
- `XR_TYPE_SWAPCHAIN_IMAGE_D3D11_KHR`

## New Enums

## New Structures

The following structures are provided to supply supporting runtimes the necessary information required to work with the D3D11 API executing on certain operating systems.

The [XrGraphicsBindingD3D11KHR](#) structure is defined as:

```
// Provided by XR_KHR_D3D11_enable
typedef struct XrGraphicsBindingD3D11KHR {
    XrStructureType    type;
    const void*        next;
    ID3D11Device*      device;
} XrGraphicsBindingD3D11KHR;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `device` is a pointer to a valid `ID3D11Device` to use.

When creating a D3D11-backed `XrSession`, the application will provide a pointer to an `XrGraphicsBindingD3D11KHR` in the `XrSessionCreateInfo::next` field of structure passed to `xrCreateSession`. The D3D11 device specified in `XrGraphicsBindingD3D11KHR::device` **must** be created in accordance with the requirements retrieved through `xrGetD3D11GraphicsRequirementsKHR`, otherwise `xrCreateSession` **must** return `XR_ERROR_GRAPHICS_DEVICE_INVALID`.

## Valid Usage (Implicit)

- The `XR_KHR_D3D11_enable` extension **must** be enabled prior to using `XrGraphicsBindingD3D11KHR`
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_D3D11_KHR`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `device` **must** be a pointer to an `ID3D11Device` value

The `XrSwapchainImageD3D11KHR` structure is defined as:

```
// Provided by XR_KHR_D3D11_enable
typedef struct XrSwapchainImageD3D11KHR {
    XrStructureType    type;
    void*              next;
    ID3D11Texture2D*   texture;
} XrSwapchainImageD3D11KHR;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `texture` is a pointer to a valid `ID3D11Texture2D` to use.

If a given session was created with [XrGraphicsBindingD3D11KHR](#), the following conditions **must** apply.

- Calls to [xrEnumerateSwapchainImages](#) on an [XrSwapchain](#) in that session **must** return an array of [XrSwapchainImageD3D11KHR](#) structures.
- Whenever an OpenXR function accepts an [XrSwapchainImageBaseHeader](#) pointer as a parameter in that session, the runtime **must** also accept a pointer to an [XrSwapchainImageD3D11KHR](#).

The OpenXR runtime **must** interpret the top-left corner of the swapchain image as the coordinate origin unless specified otherwise by extension functionality.

The OpenXR runtime **must** interpret the swapchain images in a clip space of positive Y pointing up, near Z plane at 0, and far Z plane at 1.

### Valid Usage (Implicit)

- The [XR\\_KHR\\_D3D11\\_enable](#) extension **must** be enabled prior to using [XrSwapchainImageD3D11KHR](#)
- **type** **must** be [XR\\_TYPE\\_SWAPCHAIN\\_IMAGE\\_D3D11\\_KHR](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

The [XrGraphicsRequirementsD3D11KHR](#) structure is defined as:

```
// Provided by XR_KHR_D3D11_enable
typedef struct XrGraphicsRequirementsD3D11KHR {
    XrStructureType    type;
    void*              next;
    LUID                adapterLuid;
    D3D_FEATURE_LEVEL  minFeatureLevel;
} XrGraphicsRequirementsD3D11KHR;
```

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is [NULL](#) or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **adapterLuid** identifies what graphics device needs to be used.
- **minFeatureLevel** is the minimum feature level that the D3D11 device must be initialized with.

[XrGraphicsRequirementsD3D11KHR](#) is populated by [xrGetD3D11GraphicsRequirementsKHR](#).

## Valid Usage (Implicit)

- The `XR_KHR_D3D11_enable` extension **must** be enabled prior to using `XrGraphicsRequirementsD3D11KHR`
- `type` **must** be `XR_TYPE_GRAPHICS_REQUIREMENTS_D3D11_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `adapterLuid` **must** be a valid `LUID` value
- `minFeatureLevel` **must** be a valid `D3D_FEATURE_LEVEL` value

## New Functions

Some computer systems may have multiple graphics devices, each of which may have independent external display outputs. XR systems that connect to such graphics devices are typically connected to a single device. Applications need to know what graphics device the XR system is connected to so that they can use that graphics device to generate XR images.

To retrieve the D3D11 feature level and graphics device for an instance and system, call:

```
// Provided by XR_KHR_D3D11_enable
XrResult xrGetD3D11GraphicsRequirementsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    XrGraphicsRequirementsD3D11KHR* graphicsRequirements);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `graphicsRequirements` is the `XrGraphicsRequirementsD3D11KHR` output structure.

The `xrGetD3D11GraphicsRequirementsKHR` function identifies to the application what graphics device (Windows LUID) needs to be used and the minimum feature level to use. The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` (`XR_ERROR_VALIDATION_FAILURE` **may** be returned due to legacy behavior) on calls to `xrCreateSession` if `xrGetD3D11GraphicsRequirementsKHR` has not been called for the same `instance` and `systemId`. The LUID and feature level that `xrGetD3D11GraphicsRequirementsKHR` returns **must** be used to create the `ID3D11Device` that the application passes to `xrCreateSession` in the `XrGraphicsBindingD3D11KHR`.

## Valid Usage (Implicit)

- The `XR_KHR_D3D11_enable` extension **must** be enabled prior to calling `xrGetD3D11GraphicsRequirementsKHR`
- `instance` **must** be a valid `XrInstance` handle
- `graphicsRequirements` **must** be a pointer to an `XrGraphicsRequirementsD3D11KHR` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

## Issues

### Version History

- Revision 1, 2018-05-07 (Mark Young)
  - Initial draft
- Revision 2, 2018-06-21 (Bryce Hutchings)
  - Split `XR_KHR_D3D_enable` into `XR_KHR_D3D11_enable`
  - Rename and expand `xrGetD3DGraphicsDeviceKHR` functionality to `xrGetD3D11GraphicsRequirementsKHR`
- Revision 3, 2018-11-15 (Paul Pedriana)
  - Specified the swapchain texture coordinate origin.
- Revision 4, 2018-11-16 (Minmin Gong)
  - Specified Y direction and Z range in clip space
- Revision 5, 2020-08-06 (Bryce Hutchings)
  - Added new `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` error code



- Revision 8, 2021-09-09 (Bryce Hutchings)
  - Document mapping for [XrSwapchainUsageFlags](#)
- Revision 9, 2021-12-28 (Matthieu Bucchianeri)
  - Added missing [XR\\_ERROR\\_GRAPHICS\\_DEVICE\\_INVALID](#) error condition

## 12.13. XR\_KHR\_D3D12\_enable

### Name String

[XR\\_KHR\\_D3D12\\_enable](#)

### Extension Type

Instance extension

### Registered Extension Number

29

### Revision

9

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2020-03-18

### IP Status

No known IP claims.

### Contributors

Bryce Hutchings, Microsoft  
Paul Pedriana, Oculus  
Mark Young, LunarG  
Minmin Gong, Microsoft  
Dan Ginsburg, Valve  
Matthieu Bucchianeri, Microsoft

### Overview

This extension enables the use of the D3D12 graphics API in an OpenXR runtime. Without this extension, the OpenXR runtime may not be able to use any D3D12 swapchain images.

This extension provides the mechanisms necessary for an application to generate a valid [XrGraphicsBindingD3D12KHR](#) structure in order to create a D3D12-based [XrSession](#). Note that during this process the application is responsible for creating all the required D3D12 objects, including a

graphics device and queue to be used for rendering.

This extension also provides mechanisms for the application to interact with images acquired by calling [xrEnumerateSwapchainImages](#).

In order to expose the structures, types, and functions of this extension, you **must** define `XR_USE_GRAPHICS_API_D3D12` before including the OpenXR platform header `openxr_platform.h`, in all portions of your library or application that include it.

### Swapchain Image Resource State

When an application acquires a swapchain image by calling [xrAcquireSwapchainImage](#) in a session create using [XrGraphicsBindingD3D12KHR](#), the OpenXR runtime **must** guarantee that:

- The color rendering target image has a resource state match with `D3D12_RESOURCE_STATE_RENDER_TARGET`
- The depth rendering target image has a resource state match with `D3D12_RESOURCE_STATE_DEPTH_WRITE`
- The `ID3D12CommandQueue` specified in [XrGraphicsBindingD3D12KHR](#) can write to the image.

When an application releases a swapchain image by calling [xrReleaseSwapchainImage](#), in a session create using [XrGraphicsBindingD3D12KHR](#), the OpenXR runtime **must** interpret the image as:

- Having a resource state match with `D3D12_RESOURCE_STATE_RENDER_TARGET` if the image is a color rendering target
- Having a resource state match with `D3D12_RESOURCE_STATE_DEPTH_WRITE` if the image is a depth rendering target
- Being available for read/write on the `ID3D12CommandQueue` specified in [XrGraphicsBindingD3D12KHR](#).

The application is responsible for transitioning the swapchain image back to the resource state and queue availability that the OpenXR runtime requires. If the image is not in a resource state match with the above specifications the runtime **may** exhibit undefined behavior.

All [XrSwapchainUsageFlags](#) values passed in a session created using [XrGraphicsBindingD3D12KHR](#) **must** be interpreted as follows by the runtime, so that the returned swapchain images used by the application may be used as if they were created with the corresponding `D3D12_BIND_FLAG` flags and heap type. The runtime **may** set additional resource flags but **must** not restrict usage.

| <a href="#">XrSwapchainUsageFlagBits</a>                     | Corresponding D3D12 resource flag bits                  |
|--|---|
| <code>XR_SWAPCHAIN_USAGE_COLOR_ATTACHMENT_BIT</code>         | <code>D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET</code>    |
| <code>XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT</code> | <code>D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL</code>    |
| <code>XR_SWAPCHAIN_USAGE_UNORDERED_ACCESS_BIT</code>         | <code>D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS</code> |
| <code>XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT</code>             | <i>ignored</i>  |
| <code>XR_SWAPCHAIN_USAGE_TRANSFER_DST_BIT</code>             | <i>ignored</i>  |

| <a href="#">XrSwapchainUsageFlagBits</a>   | Corresponding D3D12 resource flag bits                |
|--|---|
| <code>XR_SWAPCHAIN_USAGE_SAMPLED_BIT</code> <b>omitted</b>   | <code>D3D12_RESOURCE_FLAG_DENY_SHADER_RESOURCE</code> |
| <code>XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT</code>   | <i>ignored</i>  |
| <code>XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR</code><br>(Added by <a href="#">XR_KHR_swapchain_usage_input_attachment_bit</a> and only available when that extension is enabled) | <i>ignored</i>  |

All D3D12 swapchain textures are created with `D3D12_HEAP_TYPE_DEFAULT` usage.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_GRAPHICS_REQUIREMENTS_D3D12_KHR`
- `XR_TYPE_GRAPHICS_BINDING_D3D12_KHR`
- `XR_TYPE_SWAPCHAIN_IMAGE_D3D12_KHR`

## New Enums

## New Structures

The following structures are provided to supply supporting runtimes the necessary information required to work with the D3D12 API executing on certain operating systems.

The [XrGraphicsBindingD3D12KHR](#) structure is defined as:

```
// Provided by XR_KHR_D3D12_enable
typedef struct XrGraphicsBindingD3D12KHR {
    XrStructureType    type;
    const void*        next;
    ID3D12Device*      device;
    ID3D12CommandQueue* queue;
} XrGraphicsBindingD3D12KHR;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `device` is a pointer to a valid `ID3D12Device` to use.
- `queue` is a pointer to a valid `ID3D12CommandQueue` to use.

When creating a D3D12-backed `XrSession`, the application will provide a pointer to an `XrGraphicsBindingD3D12KHR` in the `XrSessionCreateInfo::next` field of structure passed to `xrCreateSession`. The D3D12 device specified in `XrGraphicsBindingD3D12KHR::device` **must** be created in accordance with the requirements retrieved through `xrGetD3D12GraphicsRequirementsKHR`, otherwise `xrCreateSession` **must** return `XR_ERROR_GRAPHICS_DEVICE_INVALID`.

## Valid Usage (Implicit)

- The `XR_KHR_D3D12_enable` extension **must** be enabled prior to using `XrGraphicsBindingD3D12KHR`
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_D3D12_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `device` **must** be a pointer to an `ID3D12Device` value
- `queue` **must** be a pointer to an `ID3D12CommandQueue` value

The `XrSwapchainImageD3D12KHR` structure is defined as:

```
// Provided by XR_KHR_D3D12_enable
typedef struct XrSwapchainImageD3D12KHR {
    XrStructureType    type;
    void*              next;
    ID3D12Resource*    texture;
} XrSwapchainImageD3D12KHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `texture` is a pointer to a valid [ID3D12Texture2D](#) to use.

If a given session was created with [XrGraphicsBindingD3D12KHR](#), the following conditions **must** apply.

- Calls to [xrEnumerateSwapchainImages](#) on an [XrSwapchain](#) in that session **must** return an array of [XrSwapchainImageD3D12KHR](#) structures.
- Whenever an OpenXR function accepts an [XrSwapchainImageBaseHeader](#) pointer as a parameter in that session, the runtime **must** also accept a pointer to an [XrSwapchainImageD3D12KHR](#).

The OpenXR runtime **must** interpret the top-left corner of the swapchain image as the coordinate origin unless specified otherwise by extension functionality.

The OpenXR runtime **must** interpret the swapchain images in a clip space of positive Y pointing up, near Z plane at 0, and far Z plane at 1.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_D3D12\\_enable](#) extension **must** be enabled prior to using [XrSwapchainImageD3D12KHR](#)
- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_D3D12_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrGraphicsRequirementsD3D12KHR](#) structure is defined as:

```
// Provided by XR_KHR_D3D12_enable
typedef struct XrGraphicsRequirementsD3D12KHR {
    XrStructureType    type;
    void*              next;
    LUID                adapterLuid;
    D3D_FEATURE_LEVEL  minFeatureLevel;
} XrGraphicsRequirementsD3D12KHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `adapterLuid` identifies what graphics device needs to be used.
- `minFeatureLevel` is the minimum feature level that the D3D12 device must be initialized with.

[XrGraphicsRequirementsD3D12KHR](#) is populated by [xrGetD3D12GraphicsRequirementsKHR](#).

## Valid Usage (Implicit)

- The [XR\\_KHR\\_D3D12\\_enable](#) extension **must** be enabled prior to using [XrGraphicsRequirementsD3D12KHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_REQUIREMENTS_D3D12_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `adapterLuid` **must** be a valid `LUID` value
- `minFeatureLevel` **must** be a valid `D3D_FEATURE_LEVEL` value

## New Functions

Some computer systems may have multiple graphics devices, each of which may have independent external display outputs. XR systems that connect to such graphics devices are typically connected to a single device. Applications need to know what graphics device the XR system is connected to so that they can use that graphics device to generate XR images.

To retrieve the D3D12 feature level and graphics device for an instance and system, call:

```
// Provided by XR_KHR_D3D12_enable
XrResult xrGetD3D12GraphicsRequirementsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    XrGraphicsRequirementsD3D12KHR* graphicsRequirements);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `graphicsRequirements` is the `XrGraphicsRequirementsD3D12KHR` output structure.

The `xrGetD3D12GraphicsRequirementsKHR` function identifies to the application what graphics device (Windows LUID) needs to be used and the minimum feature level to use. The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` (`XR_ERROR_VALIDATION_FAILURE` **may** be returned due to legacy behavior) on calls to `xrCreateSession` if `xrGetD3D12GraphicsRequirementsKHR` has not been called for the same `instance` and `systemId`. The LUID and feature level that `xrGetD3D12GraphicsRequirementsKHR` returns **must** be used to create the `ID3D12Device` that the application passes to `xrCreateSession` in the `XrGraphicsBindingD3D12KHR`.

## Valid Usage (Implicit)

- The `XR_KHR_D3D12_enable` extension **must** be enabled prior to calling `xrGetD3D12GraphicsRequirementsKHR`
- `instance` **must** be a valid `XrInstance` handle
- `graphicsRequirements` **must** be a pointer to an `XrGraphicsRequirementsD3D12KHR` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

## Issues

## Version History

- Revision 1, 2018-05-07 (Mark Young)
  - Initial draft
- Revision 2, 2018-06-21 (Bryce Hutchings)
  - Split `XR_KHR_D3D_enable` into `XR_KHR_D3D12_enable`
  - Rename and expand `xrGetD3DGraphicsDeviceKHR` functionality to `xrGetD3D12GraphicsRequirementsKHR`
- Revision 3, 2018-11-15 (Paul Pedriana)
  - Specified the swapchain texture coordinate origin.
- Revision 4, 2018-11-16 (Minmin Gong)
  - Specified Y direction and Z range in clip space
- Revision 5, 2019-01-29 (Dan Ginsburg)
  - Added swapchain image resource state details.
- Revision 6, 2020-03-18 (Minmin Gong)
  - Specified depth swapchain image resource state.
- Revision 7, 2020-08-06 (Bryce Hutchings)
  - Added new `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` error code
- Revision 8, 2021-09-09 (Bryce Hutchings)
  - Document mapping for `XrSwapchainUsageFlags`
- Revision 9, 2021-12-28 (Matthieu Bucchianeri)
  - Added missing `XR_ERROR_GRAPHICS_DEVICE_INVALID` error condition

## 12.14. XR\_KHR\_loader\_init

### Name String

`XR_KHR_loader_init`

### Extension Type

Instance extension

### Registered Extension Number

89

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)



## Last Modified Date

2023-05-08

## IP Status

No known IP claims.

## Contributors

Cass Everitt, Facebook

Robert Blenkinsopp, Ultraleap

## Overview

On some platforms, before loading can occur the loader must be initialized with platform-specific parameters.

Unlike other extensions, the presence of this extension is signaled by a successful call to [xrGetInstanceProcAddr](#) to retrieve the function pointer for [xrInitializeLoaderKHR](#) using [XR\\_NULL\\_HANDLE](#) as the `instance` parameter.

If this extension is supported, its use **may** be required on some platforms and the use of the [xrInitializeLoaderKHR](#) function **must** precede other OpenXR calls except [xrGetInstanceProcAddr](#).

This function exists as part of the loader library that the application is using and the loader **must** pass calls to [xrInitializeLoaderKHR](#) to the active runtime, and all enabled API layers that expose a [xrInitializeLoaderKHR](#) function exposed either through their manifest, or through their implementation of [xrGetInstanceProcAddr](#).

If the [xrInitializeLoaderKHR](#) function is discovered through the manifest, [xrInitializeLoaderKHR](#) will be called before [xrNegotiateLoaderRuntimeInterface](#) or [xrNegotiateLoaderApiLayerInterface](#) has been called on the runtime or layer respectively.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

The [XrLoaderInitInfoBaseHeaderKHR](#) structure is defined as:

```
// Provided by XR_KHR_loader_init
typedef struct XrLoaderInitInfoBaseHeaderKHR {
    XrStructureType    type;
    const void*        next;
} XrLoaderInitInfoBaseHeaderKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_loader\\_init](#) extension **must** be enabled prior to using [XrLoaderInitInfoBaseHeaderKHR](#)
- `type` **must** be `XR_TYPE_LOADER_INIT_INFO_ANDROID_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

To initialize an OpenXR loader with platform or implementation-specific parameters, call:

```
// Provided by XR_KHR_loader_init
XrResult xrInitializeLoaderKHR(
    const XrLoaderInitInfoBaseHeaderKHR* loaderInitInfo);
```

## Parameter Descriptions

- `loaderInitInfo` is a pointer to an [XrLoaderInitInfoBaseHeaderKHR](#) structure, which is a polymorphic type defined by other platform- or implementation-specific extensions.

## Issues

## Version History

- Revision 2, 2023-05-08 (Robert Blenkinsopp)
  - Explicitly state that the call to [xrInitializeLoaderKHR](#) should be passed to the runtime and enabled API layers.
- Revision 1, 2020-05-07 (Cass Everitt)
  - Initial draft

## 12.15. XR\_KHR\_loader\_init\_android

### Name String

[XR\\_KHR\\_loader\\_init\\_android](#)

### Extension Type

Instance extension

### Registered Extension Number

90

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_KHR\\_loader\\_init](#)

### Last Modified Date

2020-05-07

### IP Status

No known IP claims.

### Contributors

Cass Everitt, Facebook

### Overview

On Android, some loader implementations need the application to provide additional information on initialization. This extension defines the parameters needed by such implementations. If this is available on a given implementation, an application **must** make use of it.

On implementations where use of this is required, the following condition **must** apply:

- Whenever an OpenXR function accepts an [XrLoaderInitInfoBaseHeaderKHR](#) pointer, the runtime (and loader) **must** also accept a pointer to an [XrLoaderInitInfoAndroidKHR](#).

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_LOADER_INIT_INFO_ANDROID_KHR`

## New Enums

## New Structures

The [XrLoaderInitInfoAndroidKHR](#) structure is defined as:

```
// Provided by XR_KHR_loader_init_android
typedef struct XrLoaderInitInfoAndroidKHR {
    XrStructureType    type;
    const void*        next;
    void*               applicationVM;
    void*               applicationContext;
} XrLoaderInitInfoAndroidKHR;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `applicationVM` is a pointer to the JNI's opaque `JavaVM` structure, cast to a void pointer.
- `applicationContext` is a JNI reference to an `android.content.Context` associated with the application, cast to a void pointer.

## Valid Usage (Implicit)

- The `XR_KHR_loader_init_android` extension **must** be enabled prior to using `XrLoaderInitInfoAndroidKHR`
- `type` **must** be `XR_TYPE_LOADER_INIT_INFO_ANDROID_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `applicationVM` **must** be a pointer value
- `applicationContext` **must** be a pointer value

### New Functions

### Issues

### Version History

- Revision 1, 2020-05-07 (Cass Everitt)
  - Initial draft

## 12.16. XR\_KHR\_opengl\_enable

### Name String

`XR_KHR_opengl_enable`

### Extension Type

Instance extension

### Registered Extension Number

24

### Revision

10

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-07-02

### IP Status

No known IP claims.

## Contributors

Mark Young, LunarG  
Bryce Hutchings, Microsoft  
Paul Pedriana, Oculus  
Minmin Gong, Microsoft  
Robert Menzel, NVIDIA  
Jakob Bornecrantz, Collabora  
Paulo Gomes, Samsung Electronics

## Overview

This extension enables the use of the OpenGL graphics API in an OpenXR runtime. Without this extension, the OpenXR runtime **may** not be able to provide any OpenGL swapchain images.

This extension provides the mechanisms necessary for an application to generate a valid `XrGraphicsBindingOpenGL*KHR` structure in order to create an OpenGL-based `XrSession`. Note that during this process the application is responsible for creating an OpenGL context to be used for rendering. The runtime however will provide the OpenGL textures to render into in the form of a swapchain.

This extension provides mechanisms for the application to interact with images acquired by calling `xrEnumerateSwapchainImages`.

In order to expose the structures, types, and functions of this extension, the application **must** define `XR_USE_GRAPHICS_API_OPENGL`, as well as an appropriate `window system define` supported by this extension, before including the OpenXR platform header `openxr_platform.h`, in all portions of the library or application that include it. The window system defines currently supported by this extension are:

- `XR_USE_PLATFORM_WIN32`
- `XR_USE_PLATFORM_XLIB`
- `XR_USE_PLATFORM_XCB`
- `XR_USE_PLATFORM_WAYLAND`

Note that a runtime implementation of this extension is only required to support the structs introduced by this extension which belong to the platform it is running on.

Note that the OpenGL context given to the call `xrCreateSession` **must** not be bound in another thread when calling the functions: `xrCreateSession`, `xrDestroySession`, `xrBeginFrame`, `xrEndFrame`, `xrCreateSwapchain`, `xrDestroySwapchain`, `xrEnumerateSwapchainImages`, `xrAcquireSwapchainImage`, `xrWaitSwapchainImage` and `xrReleaseSwapchainImage`. It **may** be bound in the thread calling those functions. The runtime **must** not access the context from any other function. In particular the application must be able to call `xrWaitFrame` from a different thread than the rendering thread.

## Swapchain Flag Bits

All [XrSwapchainUsageFlags](#) valid values passed in a session created using [XrGraphicsBindingOpenGLWin32KHR](#), [XrGraphicsBindingOpenGLXlibKHR](#), [XrGraphicsBindingOpenGLXcbKHR](#) or [XrGraphicsBindingOpenGLWaylandKHR](#) **should** be ignored as there is no mapping to OpenGL texture settings.

*Note*



In such a session, a runtime **may** use a supporting graphics API, such as Vulkan, to allocate images that are intended to alias with OpenGL textures, and be part of an [XrSwapchain](#). A runtime which allocates the texture with a different graphics API **may** need to enable several usage flags on the underlying native texture resource to ensure compatibility with OpenGL.

## New Object Types

### New Flag Types

### New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_GRAPHICS_REQUIREMENTS_OPENGL_KHR`
- `XR_TYPE_GRAPHICS_BINDING_OPENGL_WIN32_KHR`
- `XR_TYPE_GRAPHICS_BINDING_OPENGL_XLIB_KHR`
- `XR_TYPE_GRAPHICS_BINDING_OPENGL_XCB_KHR`
- `XR_TYPE_GRAPHICS_BINDING_OPENGL_WAYLAND_KHR`
- `XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_KHR`

### New Enums

### New Structures

The following structures are provided to supply supporting runtimes the necessary information required to work with the OpenGL API executing on certain operating systems.

These structures are only available when the corresponding `XR_USE_PLATFORM_` macro is defined before including `openxr_platform.h`.

The [XrGraphicsBindingOpenGLWin32KHR](#) structure is defined as:

```
// Provided by XR_KHR_opengl_enable
typedef struct XrGraphicsBindingOpenGLWin32KHR {
    XrStructureType    type;
    const void*        next;
    HDC                 hDC;
    HGLRC               hGLRC;
} XrGraphicsBindingOpenGLWin32KHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `hDC` is a valid Windows HW device context handle.
- `hGLRC` is a valid Windows OpenGL rendering context handle.

When creating an OpenGL-backed [XrSession](#) on Microsoft Windows, the application will provide a pointer to an [XrGraphicsBindingOpenGLWin32KHR](#) in the `next` chain of the [XrSessionCreateInfo](#). As no standardized way exists for OpenGL to create the graphics context on a specific GPU, the runtime **must** assume that the application uses the operating systems default GPU. If the GPU used by the runtime does not match the GPU on which the OpenGL context of the application got created, [xrCreateSession](#) **must** return `XR_ERROR_GRAPHICS_DEVICE_INVALID`.

The required window system configuration define to expose this structure type is [XR\\_USE\\_PLATFORM\\_WIN32](#).

## Valid Usage (Implicit)

- The [XR\\_KHR\\_opengl\\_enable](#) extension **must** be enabled prior to using [XrGraphicsBindingOpenGLWin32KHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_OPENGL_WIN32_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `hDC` **must** be a valid `HDC` value
- `hGLRC` **must** be a valid `HGLRC` value

The [XrGraphicsBindingOpenGLLibKHR](#) structure is defined as:



```
// Provided by XR_KHR_opengl_enable
typedef struct XrGraphicsBindingOpenGlibKHR {
    XrStructureType    type;
    const void*        next;
    Display*           xDisplay;
    uint32_t           visualid;
    GLXFBConfig        glxFBConfig;
    GLXDrawable        glxDrawable;
    GLXContext         glxContext;
} XrGraphicsBindingOpenGlibKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `xDisplay` is a valid X11 [Display](#).
- `visualid` is a valid X11 visual identifier.
- `glxFBConfig` is a valid X11 OpenGL GLX [GLXFBConfig](#).
- `glxDrawable` is a valid X11 OpenGL GLX [GLXDrawable](#).
- `glxContext` is a valid X11 OpenGL GLX [GLXContext](#).

When creating an OpenGL-backed [XrSession](#) on any Linux/Unix platform that utilizes X11 and GLX, via the Xlib library, the application will provide a pointer to an [XrGraphicsBindingOpenGlibKHR](#) in the `next` chain of the [XrSessionCreateInfo](#).

The required window system configuration define to expose this structure type is [XR\\_USE\\_PLATFORM\\_XLIB](#).

## Valid Usage (Implicit)

- The `XR_KHR_opengl_enable` extension **must** be enabled prior to using `XrGraphicsBindingOpenGLESlibKHR`
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_OPENGL_XLIB_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `xDisplay` **must** be a pointer to a `Display` value
- `glxFBConfig` **must** be a valid `GLxFBConfig` value
- `glxDrawable` **must** be a valid `GLXDrawable` value
- `glxContext` **must** be a valid `GLXContext` value

The `XrGraphicsBindingOpenGLEScbKHR` structure is defined as:

```
// Provided by XR_KHR_opengl_enable
typedef struct XrGraphicsBindingOpenGLEScbKHR {
    XrStructureType    type;
    const void*        next;
    xcb_connection_t*  connection;
    uint32_t           screenNumber;
    xcb_glx_fbconfig_t fbconfigid;
    xcb_visualid_t     visualid;
    xcb_glx_drawable_t glxDrawable;
    xcb_glx_context_t  glxContext;
} XrGraphicsBindingOpenGLEScbKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `connection` is a valid `xcb_connection_t`.
- `screenNumber` is an index indicating which screen should be used for rendering.
- `fbconfigid` is a valid XCB OpenGL GLX `xcb_glx_fbconfig_t`.
- `visualid` is a valid XCB OpenGL GLX `xcb_visualid_t`.
- `glxDrawable` is a valid XCB OpenGL GLX `xcb_glx_drawable_t`.
- `glxContext` is a valid XCB OpenGL GLX `xcb_glx_context_t`.

When creating an OpenGL-backed [XrSession](#) on any Linux/Unix platform that utilizes X11 and GLX, via the Xlib library, the application will provide a pointer to an [XrGraphicsBindingOpenGLXcbKHR](#) in the `next` chain of the [XrSessionCreateInfo](#).

The required window system configuration define to expose this structure type is [XR\\_USE\\_PLATFORM\\_XCB](#).

## Valid Usage (Implicit)

- The [XR\\_KHR\\_opengl\\_enable](#) extension **must** be enabled prior to using [XrGraphicsBindingOpenGLXcbKHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_OPENGL_XCB_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `connection` **must** be a pointer to an `xcb_connection_t` value
- `fbconfigid` **must** be a valid `xcb_glx_fbconfig_t` value
- `visualid` **must** be a valid `xcb_visualid_t` value
- `glxDrawable` **must** be a valid `xcb_glx_drawable_t` value
- `glxContext` **must** be a valid `xcb_glx_context_t` value

The [XrGraphicsBindingOpenGLWaylandKHR](#) structure is defined as:

```
// Provided by XR_KHR_opengl_enable
typedef struct XrGraphicsBindingOpenGLWaylandKHR {
    XrStructureType    type;
    const void*        next;
    struct wl_display* display;
} XrGraphicsBindingOpenGLWaylandKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `display` is a valid Wayland `wl_display`.

When creating an OpenGL-backed [XrSession](#) on any Linux/Unix platform that utilizes the Wayland protocol with its compositor, the application will provide a pointer to an [XrGraphicsBindingOpenGLWaylandKHR](#) in the `next` chain of the [XrSessionCreateInfo](#).

The required window system configuration define to expose this structure type is [XR\\_USE\\_PLATFORM\\_WAYLAND](#).

## Valid Usage (Implicit)

- The [XR\\_KHR\\_opengl\\_enable](#) extension **must** be enabled prior to using [XrGraphicsBindingOpenGLWaylandKHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_OPENGL_WAYLAND_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `display` **must** be a pointer to a `wl_display` value

The [XrSwapchainImageOpenGLKHR](#) structure is defined as:

```
// Provided by XR_KHR_opengl_enable
typedef struct XrSwapchainImageOpenGLKHR {
    XrStructureType    type;
    void*              next;
    uint32_t           image;
} XrSwapchainImageOpenGLKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `image` is the OpenGL texture handle associated with this swapchain image.

If a given session was created with a [XrGraphicsBindingOpenGL\\*KHR](#), the following conditions **must** apply.

- Calls to [xrEnumerateSwapchainImages](#) on an [XrSwapchain](#) in that session **must** return an array of [XrSwapchainImageOpenGLKHR](#) structures.
- Whenever an OpenXR function accepts an [XrSwapchainImageBaseHeader](#) pointer as a parameter in that session, the runtime **must** also accept a pointer to an [XrSwapchainImageOpenGLKHR](#).

The OpenXR runtime **must** interpret the bottom-left corner of the swapchain image as the coordinate origin unless specified otherwise by extension functionality.

The OpenXR runtime **must** interpret the swapchain images in a clip space of positive Y pointing up, near Z plane at -1, and far Z plane at 1.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_opengl\\_enable](#) extension **must** be enabled prior to using [XrSwapchainImageOpenGLKHR](#)
- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrGraphicsRequirementsOpenGLKHR](#) structure is defined as:

```
// Provided by XR_KHR_opengl_enable
typedef struct XrGraphicsRequirementsOpenGLKHR {
    XrStructureType    type;
    void*              next;
    XrVersion          minApiVersionSupported;
    XrVersion          maxApiVersionSupported;
} XrGraphicsRequirementsOpenGLKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `minApiVersionSupported` is the minimum version of OpenGL that the runtime supports. Uses [XR\\_MAKE\\_VERSION](#) on major and minor API version, ignoring any patch version component.
- `maxApiVersionSupported` is the maximum version of OpenGL that the runtime has been tested on and is known to support. Newer OpenGL versions might work if they are compatible. Uses [XR\\_MAKE\\_VERSION](#) on major and minor API version, ignoring any patch version component.

[XrGraphicsRequirementsOpenGLKHR](#) is populated by [xrGetOpenGLGraphicsRequirementsKHR](#) with the runtime's OpenGL API version requirements.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_opengl\\_enable](#) extension **must** be enabled prior to using [XrGraphicsRequirementsOpenGLKHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_REQUIREMENTS_OPENGL_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

To query OpenGL API version requirements for an instance and system, call:

```
// Provided by XR_KHR_opengl_enable
XrResult xrGetOpenGLGraphicsRequirementsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    XrGraphicsRequirementsOpenGLKHR* graphicsRequirements);
```

## Parameter Descriptions

- `instance` is an [XrInstance](#) handle previously created with [xrCreateInstance](#).
- `systemId` is an [XrSystemId](#) handle for the system which will be used to create a session.
- `graphicsRequirements` is the [XrGraphicsRequirementsOpenGLKHR](#) output structure.

The `xrGetOpenGLGraphicsRequirementsKHR` function identifies to the application the minimum OpenGL version requirement and the highest known tested OpenGL version. The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` (`XR_ERROR_VALIDATION_FAILURE` **may** be returned due to legacy behavior) on calls to `xrCreateSession` if `xrGetOpenGLGraphicsRequirementsKHR` has not been called for the same `instance` and `systemId`.

### Valid Usage (Implicit)

- The `XR_KHR_opengl_enable` extension **must** be enabled prior to calling `xrGetOpenGLGraphicsRequirementsKHR`
- `instance` **must** be a valid `XrInstance` handle
- `graphicsRequirements` **must** be a pointer to an `XrGraphicsRequirementsOpenGLKHR` structure

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

### Issues

#### Version History

- Revision 1, 2018-05-07 (Mark Young)
  - Initial draft
- Revision 2, 2018-06-21 (Bryce Hutchings)
  - Add new `xrGetOpenGLGraphicsRequirementsKHR`
- Revision 3, 2018-11-15 (Paul Pedriana)
  - Specified the swapchain texture coordinate origin.
- Revision 4, 2018-11-16 (Minmin Gong)
  - Specified Y direction and Z range in clip space

- Revision 5, 2019-01-25 (Robert Menzel)
  - Description updated
- Revision 6, 2019-07-02 (Robert Menzel)
  - Minor fixes
- Revision 7, 2019-07-08 (Rylie Pavlik)
  - Adjusted member name in XCB struct
- Revision 8, 2019-11-28 (Jakob Bornecrantz)
  - Added note about context not allowed to be current in a different thread.
- Revision 9, 2020-08-06 (Bryce Hutchings)
  - Added new `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` error code
- Revision 10, 2021-08-31 (Paulo F. Gomes)
  - Document handling of `XrSwapchainUsageFlags`

## 12.17. XR\_KHR\_opengl\_es\_enable

### Name String

`XR_KHR_opengl_es_enable`

### Extension Type

Instance extension

### Registered Extension Number

25

### Revision

8

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-07-12

### IP Status

No known IP claims.

### Contributors

Mark Young, LunarG

Bryce Hutchings, Microsoft

Paul Pedriana, Oculus



Minmin Gong, Microsoft  
Robert Menzel, NVIDIA  
Martin Renschler, Qualcomm  
Paulo Gomes, Samsung Electronics

## Overview

This extension must be provided by runtimes supporting applications using OpenGL ES APIs for rendering. OpenGL ES applications need this extension to obtain compatible swapchain images which the runtime is required to supply. The runtime needs the following OpenGL ES objects from the application in order to interact properly with the OpenGL ES driver: EGLDisplay, EGLConfig and EGLContext.

These are passed from the application to the runtime in a [XrGraphicsBindingOpenGLESAndroidKHR](#) structure when creating the [XrSession](#). Although not restricted to Android, the OpenGL ES extension is currently tailored for Android.

Note that the application is responsible for creating the required OpenGL ES objects, including an OpenGL ES context to be used for rendering.

This extension also provides mechanisms for the application to interact with images acquired by calling [xrEnumerateSwapchainImages](#).

In order to expose the structures, types, and functions of this extension, the application source code **must** define [XR\\_USE\\_GRAPHICS\\_API\\_OPENGL\\_ES](#), as well as an appropriate [window system define](#), before including the OpenXR platform header `openxr_platform.h`, in all portions of your library or application that include it. The only window system define currently supported by this extension is:

- [XR\\_USE\\_PLATFORM\\_ANDROID](#)

## Swapchain Flag Bits

All [XrSwapchainUsageFlags](#) valid values passed in a session created using [XrGraphicsBindingOpenGLESAndroidKHR](#) **should** be ignored as there is no mapping to OpenGL ES texture settings.



### Note

In such a session, a runtime **may** use a supporting graphics API, such as Vulkan, to allocate images that are intended to alias with OpenGL ES textures, and be part of an [XrSwapchain](#). A runtime which allocates the texture with a different graphics API **may** need to enable several usage flags on the underlying native texture resource to ensure compatibility with OpenGL ES.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_GRAPHICS_REQUIREMENTS_OPENGL_ES_KHR`
- `XR_TYPE_GRAPHICS_BINDING_OPENGL_ES_ANDROID_KHR`
- `XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_ES_KHR`

## New Enums

### New Structures

The following structures are provided to supply supporting runtimes the necessary information required to work with the OpenGL ES API executing on certain operating systems.

These structures are only available when the corresponding `XR_USE_PLATFORM_` macro is defined before including `openxr_platform.h`.

The [XrGraphicsBindingOpenGLESAndroidKHR](#) structure is defined as:

```
// Provided by XR_KHR_opengl_es_enable
typedef struct XrGraphicsBindingOpenGLESAndroidKHR {
    XrStructureType    type;
    const void*        next;
    EGLDisplay          display;
    EGLConfig           config;
    EGLContext          context;
} XrGraphicsBindingOpenGLESAndroidKHR;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `display` is a valid Android OpenGL ES `EGLDisplay`.
- `config` is a valid Android OpenGL ES `EGLConfig`.
- `context` is a valid Android OpenGL ES `EGLContext`.

When creating an OpenGL ES-backed [XrSession](#) on Android, the application will provide a pointer to an [XrGraphicsBindingOpenGLESAndroidKHR](#) structure in the `next` chain of the [XrSessionCreateInfo](#).

The required window system configuration define to expose this structure type is [XR\\_USE\\_PLATFORM\\_ANDROID](#).

### Valid Usage (Implicit)

- The [XR\\_KHR\\_opengl\\_es\\_enable](#) extension **must** be enabled prior to using [XrGraphicsBindingOpenGLESAndroidKHR](#)
- **type** **must** be [XR\\_TYPE\\_GRAPHICS\\_BINDING\\_OPENGL\\_ES\\_ANDROID\\_KHR](#)
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **display** **must** be a valid [EGLDisplay](#) value
- **config** **must** be a valid [EGLConfig](#) value
- **context** **must** be a valid [EGLContext](#) value

The [XrSwapchainImageOpenGLESKHR](#) structure is defined as:

```
// Provided by XR_KHR_opengl_es_enable
typedef struct XrSwapchainImageOpenGLESKHR {
    XrStructureType    type;
    void*              next;
    uint32_t           image;
} XrSwapchainImageOpenGLESKHR;
```

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **image** is an index indicating the current OpenGL ES swapchain image to use.

If a given session was created with a [XrGraphicsBindingOpenGLES\\*KHR](#), the following conditions **must** apply.

- Calls to [xrEnumerateSwapchainImages](#) on an [XrSwapchain](#) in that session **must** return an array of [XrSwapchainImageOpenGLESKHR](#) structures.
- Whenever an OpenXR function accepts an [XrSwapchainImageBaseHeader](#) pointer as a parameter in that session, the runtime **must** also accept a pointer to an [XrSwapchainImageOpenGLESKHR](#) structure.

The OpenXR runtime **must** interpret the bottom-left corner of the swapchain image as the coordinate origin unless specified otherwise by extension functionality.

The OpenXR runtime **must** interpret the swapchain images in a clip space of positive Y pointing up, near Z plane at -1, and far Z plane at 1.

### Valid Usage (Implicit)

- The `XR_KHR_opengl_es_enable` extension **must** be enabled prior to using `XrSwapchainImageOpenGLESKHR`
- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_ES_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrGraphicsRequirementsOpenGLESKHR` structure is defined as:

```
// Provided by XR_KHR_opengl_es_enable
typedef struct XrGraphicsRequirementsOpenGLESKHR {
    XrStructureType    type;
    void*              next;
    XrVersion          minApiVersionSupported;
    XrVersion          maxApiVersionSupported;
} XrGraphicsRequirementsOpenGLESKHR;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `minApiVersionSupported` is the minimum version of OpenGL ES that the runtime supports. Uses `XR_MAKE_VERSION` on major and minor API version, ignoring any patch version component.
- `maxApiVersionSupported` is the maximum version of OpenGL ES that the runtime has been tested on and is known to support. Newer OpenGL ES versions might work if they are compatible. Uses `XR_MAKE_VERSION` on major and minor API version, ignoring any patch version component.

`XrGraphicsRequirementsOpenGLESKHR` is populated by `xrGetOpenGLESGraphicsRequirementsKHR` with the runtime's OpenGL ES API version requirements.

## Valid Usage (Implicit)

- The `XR_KHR_opengl_es_enable` extension **must** be enabled prior to using `XrGraphicsRequirementsOpenGLESKHR`
- `type` **must** be `XR_TYPE_GRAPHICS_REQUIREMENTS_OPENGL_ES_KHR`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

## New Functions

To query OpenGL ES API version requirements for an instance and system, call:

```
// Provided by XR_KHR_opengl_es_enable
XrResult xrGetOpenGLESGraphicsRequirementsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    XrGraphicsRequirementsOpenGLESKHR* graphicsRequirements);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `graphicsRequirements` is the `XrGraphicsRequirementsOpenGLESKHR` output structure.

The `xrGetOpenGLESGraphicsRequirementsKHR` function identifies to the application the minimum OpenGL ES version requirement and the highest known tested OpenGL ES version. The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` (`XR_ERROR_VALIDATION_FAILURE` **may** be returned due to legacy behavior) on calls to `xrCreateSession` if `xrGetOpenGLESGraphicsRequirementsKHR` has not been called for the same `instance` and `systemId`.

## Valid Usage (Implicit)

- The `XR_KHR_opengl_es_enable` extension **must** be enabled prior to calling `xrGetOpenGLESGraphicsRequirementsKHR`
- `instance` **must** be a valid `XrInstance` handle
- `graphicsRequirements` **must** be a pointer to an `XrGraphicsRequirementsOpenGLESKHR` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

## Issues

### Version History

- Revision 1, 2018-05-07 (Mark Young)
  - Initial draft
- Revision 2, 2018-06-21 (Bryce Hutchings)
  - Add new `xrGetOpenGLESGraphicsRequirementsKHR`
- Revision 3, 2018-11-15 (Paul Pedriana)
  - Specified the swapchain texture coordinate origin.
- Revision 4, 2018-11-16 (Minmin Gong)
  - Specified Y direction and Z range in clip space
- Revision 5, 2019-01-25 (Robert Menzel)
  - Description updated
- Revision 6, 2019-07-12 (Martin Renschler)
  - Description updated
- Revision 7, 2020-08-06 (Bryce Hutchings)
  - Added new `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` error code
- Revision 8, 2021-08-27 (Paulo F. Gomes)
  - Document handling of `XrSwapchainUsageFlags`

# 12.18. XR\_KHR\_swapchain\_usage\_input\_attachment\_bit

## Name String

`XR_KHR_swapchain_usage_input_attachment_bit`

## Extension Type

Instance extension

## Registered Extension Number

166

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-05-11

## IP Status

No known IP claims.

## Contributors

Jakob Bornecrantz, Collabora  
Rylie Pavlik, Collabora

## Overview

This extension enables an application to specify that swapchain images should be created in a way so that they can be used as input attachments. At the time of writing this bit only affects Vulkan swapchains.

## New Object Types

## New Flag Types

## New Enum Constants

[XrSwapchainUsageFlagBits](#) enumeration is extended with:

- `XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR` - indicates that the image format **may** be used as an input attachment.

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2020-07-23 (Jakob Bornecrantz)
  - Initial draft
- Revision 2, 2020-07-24 (Jakob Bornecrantz)
  - Added note about only affecting Vulkan
  - Changed from MNDX to MND
- Revision 3, 2021-05-11 (Rylie Pavlik, Collabora, Ltd.)
  - Updated for promotion from MND to KHR

# 12.19. XR\_KHR\_visibility\_mask

## Name String

`XR_KHR_visibility_mask`

## Extension Type

Instance extension

## Registered Extension Number

32

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2018-07-05

## IP Status

No known IP claims.

## Contributors

Paul Pedriana, Oculus  
Alex Turner, Microsoft



## Contacts

Paul Pedriana, Oculus

## Overview

This extension support the providing of a per-view drawing mask for applications. The primary purpose of this is to enable performance improvements that result from avoiding drawing on areas that are not visible to the user. A common occurrence in head-mounted VR hardware is that the optical system's frustum does not intersect precisely with the rectangular display it is viewing. As a result, it may be that there are parts of the display that are not visible to the user, such as the corners of the display. In such cases it would be unnecessary for the application to draw into those parts.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

[XrVisibilityMaskTypeKHR](#) identifies the different types of mask specification that is supported. The application **can** request a view mask in any of the formats identified by these types.

```
// Provided by XR_KHR_visibility_mask
typedef enum XrVisibilityMaskTypeKHR {
    XR_VISIBILITY_MASK_TYPE_HIDDEN_TRIANGLE_MESH_KHR = 1,
    XR_VISIBILITY_MASK_TYPE_VISIBLE_TRIANGLE_MESH_KHR = 2,
    XR_VISIBILITY_MASK_TYPE_LINE_LOOP_KHR = 3,
    XR_VISIBILITY_MASK_TYPE_MAX_ENUM_KHR = 0x7FFFFFFF
} XrVisibilityMaskTypeKHR;
```

## Enumerant Descriptions

- `XR_VISIBILITY_MASK_TYPE_HIDDEN_TRIANGLE_MESH_KHR` refers to a two dimensional triangle mesh on the view surface which **should** not be drawn to by the application. [XrVisibilityMaskKHR](#) refers to a set of triangles identified by vertices and vertex indices. The index count will thus be a multiple of three. The triangle vertices will be returned in counter-clockwise order as viewed from the user perspective.
- `XR_VISIBILITY_MASK_TYPE_VISIBLE_TRIANGLE_MESH_KHR` refers to a two dimensional triangle mesh on the view surface which **should** be drawn to by the application. [XrVisibilityMaskKHR](#) refers to a set of triangles identified by vertices and vertex indices. The index count will thus be a multiple of three. The triangle vertices will be returned in counter-clockwise order as viewed from the user perspective.
- `XR_VISIBILITY_MASK_TYPE_LINE_LOOP_KHR` refers to a single multi-segmented line loop on the view surface which encompasses the view area which **should** be drawn by the application. It is the border that exists between the visible and hidden meshes identified by `XR_VISIBILITY_MASK_TYPE_HIDDEN_TRIANGLE_MESH_KHR` and `XR_VISIBILITY_MASK_TYPE_VISIBLE_TRIANGLE_MESH_KHR`. The line is counter-clockwise, contiguous, and non-self crossing, with the last point implicitly connecting to the first point. There is one vertex per point, the index count will equal the vertex count, and the indices will refer to the vertices.

## New Structures

The [XrVisibilityMaskKHR](#) structure is an input/output struct which specifies the view mask.

```
// Provided by XR_KHR_visibility_mask
typedef struct XrVisibilityMaskKHR {
    XrStructureType    type;
    void*              next;
    uint32_t           vertexCapacityInput;
    uint32_t           vertexCountOutput;
    XrVector2f*        vertices;
    uint32_t           indexCapacityInput;
    uint32_t           indexCountOutput;
    uint32_t*          indices;
} XrVisibilityMaskKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `vertexCapacityInput` is the capacity of the `vertices` array, or `0` to indicate a request to retrieve the required capacity.
- `vertexCountOutput` is filled in by the runtime with the count of vertices written or the required capacity in the case that `vertexCapacityInput` or `indexCapacityInput` is insufficient.
- `vertices` is an array of vertices filled in by the runtime that specifies mask coordinates in the `z=-1` plane of the rendered view—i.e. one meter in front of the view. When rendering the mask for use in a projection layer, these vertices must be transformed by the application’s projection matrix used for the respective [XrCompositionLayerProjectionView](#).
- `indexCapacityInput` is the capacity of the `indices` array, or `0` to indicate a request to retrieve the required capacity.
- `indexCountOutput` is filled in by the runtime with the count of indices written or the required capacity in the case that `vertexCapacityInput` or `indexCapacityInput` is insufficient.
- `indices` is an array of indices filled in by the runtime, specifying the indices of the mask geometry in the `vertices` array.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_visibility\\_mask](#) extension **must** be enabled prior to using [XrVisibilityMaskKHR](#)
- `type` **must** be `XR_TYPE_VISIBILITY_MASK_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `vertexCapacityInput` is not `0`, `vertices` **must** be a pointer to an array of `vertexCapacityInput` [XrVector2f](#) structures
- If `indexCapacityInput` is not `0`, `indices` **must** be a pointer to an array of `indexCapacityInput` `uint32_t` values

The [XrEventDataVisibilityMaskChangedKHR](#) structure specifies an event which indicates that a given view mask has changed. The application **should** respond to the event by calling [xrGetVisibilityMaskKHR](#) to retrieve the updated mask. This event is per-view, so if the masks for multiple views in a configuration change then multiple instances of this event will be sent to the application, one per view.

```
// Provided by XR_KHR_visibility_mask
typedef struct XrEventDataVisibilityMaskChangedKHR {
    XrStructureType      type;
    const void*          next;
    XrSession            session;
    XrViewConfigurationType viewConfigurationType;
    uint32_t             viewIndex;
} XrEventDataVisibilityMaskChangedKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `session` is the [XrSession](#) for which the view mask has changed.
- `viewConfigurationType` is the view configuration whose mask has changed.
- `viewIndex` is the individual view within the view configuration to which the change refers.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_visibility\\_mask](#) extension **must** be enabled prior to using [XrEventDataVisibilityMaskChangedKHR](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_VISIBILITY_MASK_CHANGED_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrGetVisibilityMaskKHR](#) function is defined as:

```
// Provided by XR_KHR_visibility_mask
XrResult xrGetVisibilityMaskKHR(
    XrSession            session,
    XrViewConfigurationType viewConfigurationType,
    uint32_t             viewIndex,
    XrVisibilityMaskTypeKHR visibilityMaskType,
    XrVisibilityMaskKHR* visibilityMask);
```

## Parameter Descriptions

- `session` is an [XrSession](#) handle previously created with [xrCreateSession](#).
- `viewConfigurationType` is the view configuration from which to retrieve mask information.
- `viewIndex` is the individual view within the view configuration from which to retrieve mask information.
- `visibilityMaskType` is the type of visibility mask requested.
- `visibilityMask` is an input/output struct which specifies the view mask.

[xrGetVisibilityMaskKHR](#) retrieves the view mask for a given view. This function follows the [two-call idiom](#) for filling multiple buffers in a struct. Specifically, if either [XrVisibilityMaskKHR::vertexCapacityInput](#) or [XrVisibilityMaskKHR::indexCapacityInput](#) is `0`, the runtime **must** respond as if both fields were set to `0`, returning the vertex count and index count through [XrVisibilityMaskKHR::vertexCountOutput](#) or [XrVisibilityMaskKHR::indexCountOutput](#) respectively. If a view mask for the specified view isn't available, the returned vertex and index counts **must** be `0`.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_visibility\\_mask](#) extension **must** be enabled prior to calling [xrGetVisibilityMaskKHR](#)
- `session` **must** be a valid [XrSession](#) handle
- `viewConfigurationType` **must** be a valid [XrViewConfigurationType](#) value
- `visibilityMaskType` **must** be a valid [XrVisibilityMaskTypeKHR](#) value
- `visibilityMask` **must** be a pointer to an [XrVisibilityMaskKHR](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`

## Issues

### Version History

- Revision 1, 2018-07-05 (Paul Pedriana)
  - Initial version.
- Revision 2, 2019-07-15 (Alex Turner)
  - Adjust two-call idiom usage.

## 12.20. XR\_KHR\_vulkan\_enable

### Name String

`XR_KHR_vulkan_enable`

### Extension Type

Instance extension

### Registered Extension Number

26

### Revision

8

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2019-01-25

## IP Status

No known IP claims.

## Contributors

Mark Young, LunarG  
Paul Pedriana, Oculus  
Ed Hutchins, Oculus  
Andres Rodriguez, Valve  
Dan Ginsburg, Valve  
Bryce Hutchings, Microsoft  
Minmin Gong, Microsoft  
Robert Menzel, NVIDIA  
Paulo Gomes, Samsung Electronics

## Overview

This extension enables the use of the Vulkan graphics API in an OpenXR runtime. Without this extension, the OpenXR runtime may not be able to use any Vulkan swapchain images.

This extension provides the mechanisms necessary for an application to generate a valid [XrGraphicsBindingVulkanKHR](#) structure in order to create a Vulkan-based [XrSession](#). Note that during this process the application is responsible for creating all the required Vulkan objects.

This extension also provides mechanisms for the application to interact with images acquired by calling [xrEnumerateSwapchainImages](#).

In order to expose the structures, types, and functions of this extension, you **must** define [XR\\_USE\\_GRAPHICS\\_API\\_VULKAN](#) before including the OpenXR platform header [openxr\\_platform.h](#), in all portions of your library or application that include it.

## Initialization

Some of the requirements for creating a valid [XrGraphicsBindingVulkanKHR](#) include correct initialization of a [VkInstance](#), [VkPhysicalDevice](#), and [VkDevice](#).

A runtime **may** require that the [VkInstance](#) be initialized to a specific Vulkan API version. Additionally, the runtime **may** require a set of instance extensions to be enabled in the [VkInstance](#). These requirements can be queried by the application using [xrGetVulkanGraphicsRequirementsKHR](#) and [xrGetVulkanInstanceExtensionsKHR](#), respectively.

Similarly, the runtime **may** require the [VkDevice](#) to have a set of device extensions enabled, which can

be queried using [xrGetVulkanDeviceExtensionsKHR](#).

In order to satisfy the `VkPhysicalDevice` requirements, the application can query [xrGetVulkanGraphicsDeviceKHR](#) to identify the correct `VkPhysicalDevice`.

Populating an [XrGraphicsBindingVulkanKHR](#) with a `VkInstance`, `VkDevice`, or `VkPhysicalDevice` that does not meet the requirements outlined by this extension **may** result in undefined behavior by the OpenXR runtime.

The API version, instance extension, device extension and physical device requirements only apply to the `VkInstance`, `VkDevice`, and `VkPhysicalDevice` objects which the application wishes to associate with an [XrGraphicsBindingVulkanKHR](#).

## Concurrency

Vulkan requires that concurrent access to a `VkQueue` from multiple threads be externally synchronized. Therefore, OpenXR functions that may access the `VkQueue` specified in the [XrGraphicsBindingVulkanKHR](#) must also be externally synchronized.

The list of OpenXR functions where the OpenXR runtime **may** access the `VkQueue` are:

- [xrBeginFrame](#)
- [xrEndFrame](#)
- [xrAcquireSwapchainImage](#)
- [xrReleaseSwapchainImage](#)

The runtime **must** not access the `VkQueue` in any OpenXR function that is not listed above or in an extension definition.

## Swapchain Image Layout

When an application acquires a swapchain image by calling [xrAcquireSwapchainImage](#) in a session created using [XrGraphicsBindingVulkanKHR](#), the OpenXR runtime **must** guarantee that:

- The image has a memory layout compatible with `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` for color images, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` for depth images.
- The `VkQueue` specified in [XrGraphicsBindingVulkanKHR](#) has ownership of the image.

When an application releases a swapchain image by calling [xrReleaseSwapchainImage](#), in a session created using [XrGraphicsBindingVulkanKHR](#), the OpenXR runtime **must** interpret the image as:

- Having a memory layout compatible with `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` for color images, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` for depth images.
- Being owned by the `VkQueue` specified in [XrGraphicsBindingVulkanKHR](#).

The application is responsible for transitioning the swapchain image back to the image layout and



queue ownership that the OpenXR runtime requires. If the image is not in a layout compatible with the above specifications the runtime **may** exhibit undefined behavior.

## Swapchain Flag Bits

All [XrSwapchainUsageFlags](#) values passed in a session created using [XrGraphicsBindingVulkanKHR](#) **must** be interpreted as follows by the runtime, so that the returned swapchain images used by the application may be used as if they were created with at least the specified [VkImageUsageFlagBits](#) or [VkImageCreateFlagBits](#) set.

| <a href="#">XrSwapchainUsageFlagBits</a>   | Corresponding Vulkan flag bit                               |
|--|---|
| <a href="#">XR_SWAPCHAIN_USAGE_COLOR_ATTACHMENT_BIT</a>  | <a href="#">VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT</a>         |
| <a href="#">XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT</a>  | <a href="#">VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT</a> |
| <a href="#">XR_SWAPCHAIN_USAGE_UNORDERED_ACCESS_BIT</a>  | <a href="#">VK_IMAGE_USAGE_STORAGE_BIT</a>                  |
| <a href="#">XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT</a>  | <a href="#">VK_IMAGE_USAGE_TRANSFER_SRC_BIT</a>             |
| <a href="#">XR_SWAPCHAIN_USAGE_TRANSFER_DST_BIT</a>  | <a href="#">VK_IMAGE_USAGE_TRANSFER_DST_BIT</a>             |
| <a href="#">XR_SWAPCHAIN_USAGE_SAMPLED_BIT</a>   | <a href="#">VK_IMAGE_USAGE_SAMPLED_BIT</a>                  |
| <a href="#">XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT</a>  | <a href="#">VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT</a>          |
| <a href="#">XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR</a><br>(Added by <a href="#">XR_KHR_swapchain_usage_input_attachment_bit</a> and only available when that extension is enabled)                              | <a href="#">VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT</a>         |
| <a href="#">XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_MND</a><br>(Added by the now deprecated <a href="#">XR_MND_swapchain_usage_input_attachment_bit</a> extension and only available when that extension is enabled) | <a href="#">VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT</a>         |

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_GRAPHICS\\_REQUIREMENTS\\_VULKAN\\_KHR](#)
- [XR\\_TYPE\\_GRAPHICS\\_BINDING\\_VULKAN\\_KHR](#)
- [XR\\_TYPE\\_SWAPCHAIN\\_IMAGE\\_VULKAN\\_KHR](#)

## New Enums

## New Structures

The following structures are provided to supply supporting runtimes the necessary information required to work with the Vulkan API executing on certain operating systems.

The [XrGraphicsBindingVulkanKHR](#) structure is defined as:

```
// Provided by XR_KHR_vulkan_enable
typedef struct XrGraphicsBindingVulkanKHR {
    XrStructureType    type;
    const void*       next;
    VkInstance         instance;
    VkPhysicalDevice   physicalDevice;
    VkDevice           device;
    uint32_t          queueFamilyIndex;
    uint32_t          queueIndex;
} XrGraphicsBindingVulkanKHR;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `instance` is a valid Vulkan [VkInstance](#).
- `physicalDevice` is a valid Vulkan [VkPhysicalDevice](#).
- `device` is a valid Vulkan [VkDevice](#).
- `queueFamilyIndex` is a valid queue family index on `device`.
- `queueIndex` is a valid queue index on `device` to be used for synchronization.

When creating a Vulkan-backed [XrSession](#), the application will provide a pointer to an [XrGraphicsBindingVulkanKHR](#) in the `next` chain of the [XrSessionCreateInfo](#).

## Valid Usage

- **instance** **must** have enabled a Vulkan API version in the range specified by [XrGraphicsBindingVulkanKHR](#)
- **instance** **must** have enabled all the instance extensions specified by [xrGetVulkanInstanceExtensionsKHR](#)
- **physicalDevice** [VkPhysicalDevice](#) **must** match the device specified by [xrGetVulkanGraphicsDeviceKHR](#)
- **device** **must** have enabled all the device extensions specified by [xrGetVulkanDeviceExtensionsKHR](#)

## Valid Usage (Implicit)

- The [XR\\_KHR\\_vulkan\\_enable](#) extension **must** be enabled prior to using [XrGraphicsBindingVulkanKHR](#)
- **type** **must** be [XR\\_TYPE\\_GRAPHICS\\_BINDING\\_VULKAN\\_KHR](#)
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **instance** **must** be a valid [VkInstance](#) value
- **physicalDevice** **must** be a valid [VkPhysicalDevice](#) value
- **device** **must** be a valid [VkDevice](#) value

The [XrSwapchainImageVulkanKHR](#) structure is defined as:

```
// Provided by XR_KHR_vulkan_enable
typedef struct XrSwapchainImageVulkanKHR {
    XrStructureType    type;
    void*              next;
    VkImage             image;
} XrSwapchainImageVulkanKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `image` is a valid Vulkan [VkImage](#) to use.

If a given session was created with [XrGraphicsBindingVulkanKHR](#), the following conditions **must** apply.

- Calls to [xrEnumerateSwapchainImages](#) on an [XrSwapchain](#) in that session **must** return an array of [XrSwapchainImageVulkanKHR](#) structures.
- Whenever an OpenXR function accepts an [XrSwapchainImageBaseHeader](#) pointer as a parameter in that session, the runtime **must** also accept a pointer to an [XrSwapchainImageVulkanKHR](#).

The OpenXR runtime **must** interpret the top-left corner of the swapchain image as the coordinate origin unless specified otherwise by extension functionality.

The OpenXR runtime **must** interpret the swapchain images in a clip space of positive Y pointing down, near Z plane at 0, and far Z plane at 1.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_vulkan\\_enable](#) extension **must** be enabled prior to using [XrSwapchainImageVulkanKHR](#)
- `type` **must** be [XR\\_TYPE\\_SWAPCHAIN\\_IMAGE\\_VULKAN\\_KHR](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrSwapchainImageFoveationVulkanFB](#)

The [XrGraphicsRequirementsVulkanKHR](#) structure is defined as:

```
// Provided by XR_KHR_vulkan_enable
typedef struct XrGraphicsRequirementsVulkanKHR {
    XrStructureType    type;
    void*              next;
    XrVersion           minApiVersionSupported;
    XrVersion           maxApiVersionSupported;
} XrGraphicsRequirementsVulkanKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `minApiVersionSupported` is the minimum Vulkan Instance API version that the runtime supports. Uses [XR\\_MAKE\\_VERSION](#) on major and minor API version, ignoring any patch version component.
- `maxApiVersionSupported` is the maximum Vulkan Instance API version that the runtime has been tested on and is known to support. Newer Vulkan Instance API versions might work if they are compatible. Uses [XR\\_MAKE\\_VERSION](#) on major and minor API version, ignoring any patch version component.

[XrGraphicsRequirementsVulkanKHR](#) is populated by [xrGetVulkanGraphicsRequirementsKHR](#) with the runtime's Vulkan API version requirements.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_vulkan\\_enable](#) extension **must** be enabled prior to using [XrGraphicsRequirementsVulkanKHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

To query Vulkan API version requirements, call:

```
// Provided by XR_KHR_vulkan_enable
XrResult xrGetVulkanGraphicsRequirementsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    XrGraphicsRequirementsVulkanKHR* graphicsRequirements);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `graphicsRequirements` is the `XrGraphicsRequirementsVulkanKHR` output structure.

The `xrGetVulkanGraphicsRequirementsKHR` function identifies to the application the minimum Vulkan version requirement and the highest known tested Vulkan version. The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` (`XR_ERROR_VALIDATION_FAILURE` **may** be returned due to legacy behavior) on calls to `xrCreateSession` if `xrGetVulkanGraphicsRequirementsKHR` has not been called for the same `instance` and `systemId`.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable` extension **must** be enabled prior to calling `xrGetVulkanGraphicsRequirementsKHR`
- `instance` **must** be a valid `XrInstance` handle
- `graphicsRequirements` **must** be a pointer to an `XrGraphicsRequirementsVulkanKHR` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

Some computer systems may have multiple graphics devices, each of which may have independent external display outputs. XR systems that connect to such graphics devices are typically connected to a single device. Applications need to know what graphics device the XR system is connected to so that they can use that graphics device to generate XR images.

To identify what graphics device needs to be used for an instance and system, call:

```
// Provided by XR_KHR_vulkan_enable
XrResult xrGetVulkanGraphicsDeviceKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    VkInstance          vkInstance,
    VkPhysicalDevice*  vkPhysicalDevice);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `vkInstance` is a valid Vulkan `VkInstance`.
- `vkPhysicalDevice` is a pointer to a `VkPhysicalDevice` value to populate.

`xrGetVulkanGraphicsDeviceKHR` function identifies to the application what graphics device (Vulkan `VkPhysicalDevice`) needs to be used. `xrGetVulkanGraphicsDeviceKHR` **must** be called prior to calling `xrCreateSession`, and the `VkPhysicalDevice` that `xrGetVulkanGraphicsDeviceKHR` returns should be passed to `xrCreateSession` in the `XrGraphicsBindingVulkanKHR`.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable` extension **must** be enabled prior to calling `xrGetVulkanGraphicsDeviceKHR`
- `instance` **must** be a valid `XrInstance` handle
- `vkInstance` **must** be a valid `VkInstance` value
- `vkPhysicalDevice` **must** be a pointer to a `VkPhysicalDevice` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

```
// Provided by XR_KHR_vulkan_enable
XrResult xrGetVulkanInstanceExtensionsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    uint32_t           bufferCapacityInput,
    uint32_t*          bufferCountOutput,
    char*              buffer);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `bufferCapacityInput` is the capacity of the `buffer`, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is a pointer to the count of characters written (including terminating `\0`), or a pointer to the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an array of characters, but **can** be `NULL` if `bufferCapacityInput` is 0. The format of the output is a single space (ASCII `0x20`) delimited string of extension names.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.



## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable` extension **must** be enabled prior to calling `xrGetVulkanInstanceExtensionsKHR`
- `instance` **must** be a valid `XrInstance` handle
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not `0`, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SYSTEM_INVALID`

```
// Provided by XR_KHR_vulkan_enable
XrResult xrGetVulkanDeviceExtensionsKHR(
    XrInstance          instance,
    XrSystemId         systemId,
    uint32_t           bufferCapacityInput,
    uint32_t*         bufferCountOutput,
    char*              buffer);
```

## Parameter Descriptions

- `instance` is an [XrInstance](#) handle previously created with [xrCreateInstance](#).
- `systemId` is an [XrSystemId](#) handle for the system which will be used to create a session.
- `bufferCapacityInput` is the capacity of the `buffer`, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is a pointer to the count of characters written (including terminating `\0`), or a pointer to the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an array of characters, but **can** be `NULL` if `bufferCapacityInput` is 0. The format of the output is a single space (ASCII `0x20`) delimited string of extension names.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_vulkan\\_enable](#) extension **must** be enabled prior to calling [xrGetVulkanDeviceExtensionsKHR](#)
- `instance` **must** be a valid [XrInstance](#) handle
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SYSTEM_INVALID`

## Issues

### Version History

- Revision 1, 2018-05-07 (Mark Young)
  - Initial draft
- Revision 2, 2018-06-21 (Bryce Hutchings)
  - Replace `session` parameter with `instance` and `systemId` parameters.
  - Move `xrGetVulkanDeviceExtensionsKHR`, `xrGetVulkanInstanceExtensionsKHR` and `xrGetVulkanGraphicsDeviceKHR` functions into this extension
  - Add new `XrGraphicsRequirementsVulkanKHR` function.
- Revision 3, 2018-11-15 (Paul Pedriana)
  - Specified the swapchain texture coordinate origin.
- Revision 4, 2018-11-16 (Minmin Gong)
  - Specified Y direction and Z range in clip space
- Revision 5, 2019-01-24 (Robert Menzel)
  - Description updated
- Revision 6, 2019-01-25 (Andres Rodriguez)
  - Reword sections of the spec to shift requirements on to the runtime instead of the app
- Revision 7, 2020-08-06 (Bryce Hutchings)
  - Added new `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` error code
- Revision 8, 2021-01-21 (Rylie Pavlik, Collabora, Ltd.)
  - Document mapping for `XrSwapchainUsageFlags`

## 12.21. XR\_KHR\_vulkan\_enable2

### Name String

`XR_KHR_vulkan_enable2`

### Extension Type

Instance extension

### Registered Extension Number

91

### Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2020-05-04

## IP Status

No known IP claims.

## Contributors

Mark Young, LunarG  
Paul Pedriana, Oculus  
Ed Hutchins, Oculus  
Andres Rodriguez, Valve  
Dan Ginsburg, Valve  
Bryce Hutchings, Microsoft  
Minmin Gong, Microsoft  
Robert Menzel, NVIDIA  
Paulo Gomes, Samsung Electronics

### 12.21.1. Overview

This extension enables the use of the Vulkan graphics API in an OpenXR runtime. Without this extension, the OpenXR runtime may not be able to use any Vulkan swapchain images.

This extension provides the mechanisms necessary for an application to generate a valid [XrGraphicsBindingVulkan2KHR](#) structure in order to create a Vulkan-based [XrSession](#).

This extension also provides mechanisms for the application to interact with images acquired by calling [xrEnumerateSwapchainImages](#).

In order to expose the structures, types, and functions of this extension, you **must** define [XR\\_USE\\_GRAPHICS\\_API\\_VULKAN](#) before including the OpenXR platform header [openxr\\_platform.h](#), in all portions of your library or application that include it.



#### Note

This extension is intended as an alternative to [XR\\_KHR\\_vulkan\\_enable](#), and does not depend on it.

### 12.21.2. Initialization

When operating in Vulkan mode, the OpenXR runtime and the application will share the Vulkan queue described in the [XrGraphicsBindingVulkan2KHR](#) structure. This section of the document describes the mechanisms this extension exposes to ensure the shared Vulkan queue is compatible with the runtime and the application's requirements.

## Vulkan Version Requirements

First, a compatible Vulkan version **must** be agreed upon. To query the runtime's Vulkan API version requirements an application will call:

```
// Provided by XR_KHR_vulkan_enable2
XrResult xrGetVulkanGraphicsRequirements2KHR(
    XrInstance          instance,
    XrSystemId         systemId,
    XrGraphicsRequirementsVulkanKHR* graphicsRequirements);
```

The `xrGetVulkanGraphicsRequirements2KHR` function identifies to the application the runtime's minimum Vulkan version requirement and the highest known tested Vulkan version. `xrGetVulkanGraphicsRequirements2KHR` **must** be called prior to calling `xrCreateSession`. The runtime **must** return `XR_ERROR_GRAPHICS_REQUIREMENTS_CALL_MISSING` on calls to `xrCreateSession` if `xrGetVulkanGraphicsRequirements2KHR` has not been called for the same `instance` and `systemId`.

### Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `graphicsRequirements` is the `XrGraphicsRequirementsVulkan2KHR` output structure.

### Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to calling `xrGetVulkanGraphicsRequirements2KHR`
- `instance` **must** be a valid `XrInstance` handle
- `graphicsRequirements` **must** be a pointer to an `XrGraphicsRequirementsVulkanKHR` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

The `XrGraphicsRequirementsVulkan2KHR` structure populated by `xrGetVulkanGraphicsRequirements2KHR` is defined as:

```
// Provided by XR_KHR_vulkan_enable2
// XrGraphicsRequirementsVulkan2KHR is an alias for XrGraphicsRequirementsVulkanKHR
typedef struct XrGraphicsRequirementsVulkanKHR {
    XrStructureType    type;
    void*              next;
    XrVersion          minApiVersionSupported;
    XrVersion          maxApiVersionSupported;
} XrGraphicsRequirementsVulkanKHR;

typedef XrGraphicsRequirementsVulkanKHR XrGraphicsRequirementsVulkan2KHR;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `minApiVersionSupported` is the minimum version of Vulkan that the runtime supports. Uses `XR_MAKE_VERSION` on major and minor API version, ignoring any patch version component.
- `maxApiVersionSupported` is the maximum version of Vulkan that the runtime has been tested on and is known to support. Newer Vulkan versions might work if they are compatible. Uses `XR_MAKE_VERSION` on major and minor API version, ignoring any patch version component.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to using `XrGraphicsRequirementsVulkan2KHR`
- **Note:** `XrGraphicsRequirementsVulkan2KHR` is an alias for `XrGraphicsRequirementsVulkanKHR`, so the following items replicate the implicit valid usage for `XrGraphicsRequirementsVulkanKHR`
- **type must** be `XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN_KHR`
- **next must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## Vulkan Instance Creation

Second, a compatible `VkInstance` **must** be created. The `xrCreateVulkanInstanceKHR` entry point is a wrapper around `vkCreateInstance` intended for this purpose. When called, the runtime **must** aggregate the requirements specified by the application with its own requirements and forward the `VkInstance` creation request to the `vkCreateInstance` function pointer returned by `pfnGetInstanceProcAddr`.

```
// Provided by XR_KHR_vulkan_enable2
XrResult xrCreateVulkanInstanceKHR(
    XrInstance                instance,
    const XrVulkanInstanceCreateInfoKHR* createInfo,
    VkInstance*               vulkanInstance,
    VkResult*                  vulkanResult);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `createInfo` extensible input struct of type `XrVulkanInstanceCreateInfoKHR`
- `vulkanInstance` points to a `VkInstance` handle to populate with the new Vulkan instance.
- `vulkanResult` points to a `VkResult` to populate with the result of the `vkCreateInstance` operation as returned by `XrVulkanInstanceCreateInfoKHR::pfnGetInstanceProcAddr`.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to calling `xrCreateVulkanInstanceKHR`
- `instance` **must** be a valid `XrInstance` handle
- `createInfo` **must** be a pointer to a valid `XrVulkanInstanceCreateInfoKHR` structure
- `vulkanInstance` **must** be a pointer to a `VkInstance` value
- `vulkanResult` **must** be a pointer to a `VkResult` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_SYSTEM_INVALID`

The `XrVulkanInstanceCreateInfoKHR` structure contains the input parameters to `xrCreateVulkanInstanceKHR`.



```
// Provided by XR_KHR_vulkan_enable2
typedef struct XrVulkanInstanceCreateInfoKHR {
    XrStructureType          type;
    const void*              next;
    XrSystemId               systemId;
    XrVulkanInstanceCreateFlagsKHR createFlags;
    PFN_vkGetInstanceProcAddr pfnGetInstanceProcAddr;
    const VkInstanceCreateInfo* vulkanCreateInfo;
    const VkAllocationCallbacks* vulkanAllocator;
} XrVulkanInstanceCreateInfoKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension
- `systemId` is an [XrSystemId](#) handle for the system which will be used to create a session.
- `createFlags` is a bitmask of [XrVulkanInstanceCreateFlagBitsKHR](#)
- `pfnGetInstanceProcAddr` is a function pointer to `vkGetInstanceProcAddr` or a compatible entry point.
- `vulkanCreateInfo` is the [VkInstanceCreateInfo](#) as specified by Vulkan.
- `vulkanAllocator` is the [VkAllocationCallbacks](#) as specified by Vulkan.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_vulkan\\_enable2](#) extension **must** be enabled prior to using [XrVulkanInstanceCreateInfoKHR](#)
- `type` **must** be `XR_TYPE_VULKAN_INSTANCE_CREATE_INFO_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `createFlags` **must** be `0`
- `pfnGetInstanceProcAddr` **must** be a valid `PFN_vkGetInstanceProcAddr` value
- `vulkanCreateInfo` **must** be a pointer to a valid `VkInstanceCreateInfo` value
- If `vulkanAllocator` is not `NULL`, `vulkanAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` value

The `XrVulkanInstanceCreateInfoKHR::createFlags` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in [XrVulkanInstanceCreateFlagBitsKHR](#).

```
typedef XrFlags64 XrVulkanInstanceCreateFlagsKHR;
```

Valid bits for [XrVulkanInstanceCreateFlagsKHR](#) are defined by [XrVulkanInstanceCreateFlagBitsKHR](#).

```
// Flag bits for XrVulkanInstanceCreateFlagsKHR
```

There are currently no Vulkan instance creation flag bits defined. This is reserved for future use.

### Physical Device Selection

Third, a [VkPhysicalDevice](#) **must** be chosen. Some computer systems may have multiple graphics devices, each of which may have independent external display outputs. The runtime **must** report a [VkPhysicalDevice](#) that is compatible with the OpenXR implementation when [xrGetVulkanGraphicsDevice2KHR](#) is invoked. The application will use this [VkPhysicalDevice](#) to interact with the OpenXR runtime.

```
// Provided by XR_KHR_vulkan_enable2
XrResult xrGetVulkanGraphicsDevice2KHR(
    XrInstance instance,
    const XrVulkanGraphicsDeviceGetInfoKHR* getInfo,
    VkPhysicalDevice* vulkanPhysicalDevice);
```

### Parameter Descriptions

- [instance](#) is an [XrInstance](#) handle previously created with [xrCreateInstance](#).
- [getInfo](#) extensible input struct of type [XrVulkanGraphicsDeviceGetInfoKHR](#)
- [vulkanPhysicalDevice](#) is a pointer to a [VkPhysicalDevice](#) handle to populate.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to calling `xrGetVulkanGraphicsDevice2KHR`
- `instance` **must** be a valid `XrInstance` handle
- `getInfo` **must** be a pointer to a valid `XrVulkanGraphicsDeviceGetInfoKHR` structure
- `vulkanPhysicalDevice` **must** be a pointer to a `VkPhysicalDevice` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SYSTEM_INVALID`

The `XrVulkanGraphicsDeviceGetInfoKHR` structure contains the input parameters to `xrCreateVulkanInstanceKHR`.

```
// Provided by XR_KHR_vulkan_enable2
typedef struct XrVulkanGraphicsDeviceGetInfoKHR {
    XrStructureType    type;
    const void*        next;
    XrSystemId         systemId;
    VkInstance         vulkanInstance;
} XrVulkanGraphicsDeviceGetInfoKHR;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `systemId` is an `XrSystemId` handle for the system which will be used to create a session.
- `vulkanInstance` is a valid Vulkan `VkInstance`.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to using `XrVulkanGraphicsDeviceGetInfoKHR`
- `type` **must** be `XR_TYPE_VULKAN_GRAPHICS_DEVICE_GET_INFO_KHR`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `vulkanInstance` **must** be a valid `VkInstance` value

## Vulkan Device Creation

Fourth, a compatible `VkDevice` **must** be created. The `xrCreateVulkanDeviceKHR` entry point is a wrapper around `vkCreateDevice` intended for this purpose. When called, the runtime **must** aggregate the requirements specified by the application with its own requirements and forward the `VkDevice` creation request to the `vkCreateDevice` function pointer returned by `XrVulkanInstanceCreateInfoKHR::pfnGetInstanceProcAddr`.

```
// Provided by XR_KHR_vulkan_enable2
XrResult xrCreateVulkanDeviceKHR(
    XrInstance                instance,
    const XrVulkanDeviceCreateInfoKHR* createInfo,
    VkDevice*                vulkanDevice,
    VkResult*                vulkanResult);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `createInfo` extensible input struct of type `XrCreateVulkanDeviceCreateInfoKHR`
- `vulkanDevice` points to a `VkDevice` handle to populate with the new Vulkan device.
- `vulkanResult` points to a `VkResult` to populate with the result of the `vkCreateDevice` operation as returned by `XrVulkanInstanceCreateInfoKHR::pfnGetInstanceProcAddr`.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to calling `xrCreateVulkanDeviceKHR`
- `instance` **must** be a valid `XrInstance` handle
- `createInfo` **must** be a pointer to a valid `XrVulkanDeviceCreateInfoKHR` structure
- `vulkanDevice` **must** be a pointer to a `VkDevice` value
- `vulkanResult` **must** be a pointer to a `VkResult` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_SYSTEM_INVALID`

The `XrVulkanDeviceCreateInfoKHR` structure contains the input parameters to `xrCreateVulkanDeviceKHR`.

```
// Provided by XR_KHR_vulkan_enable2
typedef struct XrVulkanDeviceCreateInfoKHR {
    XrStructureType          type;
    const void*              next;
    XrSystemId               systemId;
    XrVulkanDeviceCreateFlagsKHR createFlags;
    PFN_vkGetInstanceProcAddr pfnGetInstanceProcAddr;
    VkPhysicalDevice          vulkanPhysicalDevice;
    const VkDeviceCreateInfo* vulkanCreateInfo;
    const VkAllocationCallbacks* vulkanAllocator;
} XrVulkanDeviceCreateInfoKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `systemId` is an [XrSystemId](#) handle for the system which will be used to create a session.
- `createFlags` is a bitmask of [XrVulkanDeviceCreateFlagBitsKHR](#)
- `pfnGetInstanceProcAddr` is a function pointer to `vkGetInstanceProcAddr` or a compatible entry point.
- `vulkanPhysicalDevice` **must** match [xrGetVulkanGraphicsDeviceKHR](#).
- `vulkanCreateInfo` is the [VkDeviceCreateInfo](#) as specified by Vulkan.
- `vulkanAllocator` is the [VkAllocationCallbacks](#) as specified by Vulkan.

If the `vulkanPhysicalDevice` parameter does not match the output of [xrGetVulkanGraphicsDeviceKHR](#), then the runtime **must** return `XR_ERROR_HANDLE_INVALID`.

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to using `XrVulkanDeviceCreateInfoKHR`
- `type` **must** be `XR_TYPE_VULKAN_DEVICE_CREATE_INFO_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `createFlags` **must** be `0`
- `pfnGetInstanceProcAddr` **must** be a valid `PFN_vkGetInstanceProcAddr` value
- `vulkanPhysicalDevice` **must** be a valid `VkPhysicalDevice` value
- `vulkanCreateInfo` **must** be a pointer to a valid `VkDeviceCreateInfo` value
- If `vulkanAllocator` is not `NULL`, `vulkanAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` value

`XrVulkanDeviceCreateFlagsKHR` specify details of device creation. The `XrVulkanDeviceCreateInfoKHR::createFlags` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in `XrVulkanDeviceCreateFlagBitsKHR`.

```
typedef XrFlags64 XrVulkanDeviceCreateFlagsKHR;
```

Valid bits for `XrVulkanDeviceCreateFlagsKHR` are defined by `XrVulkanDeviceCreateFlagBitsKHR`.

```
// Flag bits for XrVulkanDeviceCreateFlagsKHR
```

There are currently no Vulkan device creation flag bits defined. This is reserved for future use.

## Queue Selection

Last, the application selects a `VkQueue` from the `VkDevice` that has the `VK_QUEUE_GRAPHICS_BIT` set.



### Note

The runtime may schedule work on the `VkQueue` specified in the binding, or it may schedule work on any hardware queue in a foreign logical device.

## Vulkan Graphics Binding

When creating a Vulkan-backed [XrSession](#), the application will chain a pointer to an [XrGraphicsBindingVulkan2KHR](#) to the [XrSessionCreateInfo](#) parameter of [xrCreateSession](#). With the data collected in the previous sections, the application now has all the necessary information to populate an [XrGraphicsBindingVulkan2KHR](#) structure for session creation.

```
// Provided by XR_KHR_vulkan_enable2
// XrGraphicsBindingVulkan2KHR is an alias for XrGraphicsBindingVulkanKHR
typedef struct XrGraphicsBindingVulkanKHR {
    XrStructureType    type;
    const void*       next;
    VkInstance         instance;
    VkPhysicalDevice   physicalDevice;
    VkDevice           device;
    uint32_t          queueFamilyIndex;
    uint32_t          queueIndex;
} XrGraphicsBindingVulkanKHR;

typedef XrGraphicsBindingVulkanKHR XrGraphicsBindingVulkan2KHR;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `instance` is a valid Vulkan [VkInstance](#).
- `physicalDevice` is a valid Vulkan [VkPhysicalDevice](#).
- `device` is a valid Vulkan [VkDevice](#).
- `queueFamilyIndex` is a valid queue family index on `device`.
- `queueIndex` is a valid queue index on `device` to be used for synchronization.



## Valid Usage

- `instance` **must** have enabled a Vulkan API version in the range specified by [xrGetVulkanGraphicsRequirements2KHR](#)
- `instance` **must** have been created using [xrCreateVulkanInstanceKHR](#)
- `physicalDevice` `VkPhysicalDevice` **must** match the device specified by [xrGetVulkanGraphicsDevice2KHR](#)
- `device` **must** have been created using [xrCreateVulkanDeviceKHR](#)

## Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to using [XrGraphicsBindingVulkan2KHR](#)
- **Note:** [XrGraphicsBindingVulkan2KHR](#) is an alias for [XrGraphicsBindingVulkanKHR](#), so the following items replicate the implicit valid usage for [XrGraphicsBindingVulkanKHR](#)
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_VULKAN_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `instance` **must** be a valid `VkInstance` value
- `physicalDevice` **must** be a valid `VkPhysicalDevice` value
- `device` **must** be a valid `VkDevice` value

Populating an [XrGraphicsBindingVulkan2KHR](#) structure with a member that does not meet the requirements outlined by this extension **may** result in undefined behavior by the OpenXR runtime.

The requirements outlined in this extension only apply to the `VkInstance`, `VkDevice`, `VkPhysicalDevice` and `VkQueue` objects which the application wishes to associate with an [XrGraphicsBindingVulkan2KHR](#).

### 12.21.3. Concurrency

Vulkan requires that concurrent access to a `VkQueue` from multiple threads be externally synchronized. Therefore, OpenXR functions that may access the `VkQueue` specified in the [XrGraphicsBindingVulkan2KHR](#) **must** also be externally synchronized by the OpenXR application.

The list of OpenXR functions where the OpenXR runtime **may** access the `VkQueue` are:

- [xrBeginFrame](#)
- [xrEndFrame](#)
- [xrAcquireSwapchainImage](#)
- [xrReleaseSwapchainImage](#)

The runtime **must** not access the `VkQueue` in any OpenXR function that is not listed above or in an extension definition.

Failure by the application to synchronize access to `VkQueue` **may** result in undefined behavior in the OpenXR runtime.

## 12.21.4. Swapchain Interactions

### Swapchain Images

When an application interacts with `XrSwapchainImageBaseHeader` structures in a Vulkan-backed `XrSession`, the application can interpret these to be `XrSwapchainImageVulkan2KHR` structures. These are defined as:

```
// Provided by XR_KHR_vulkan_enable2
// XrSwapchainImageVulkan2KHR is an alias for XrSwapchainImageVulkanKHR
typedef struct XrSwapchainImageVulkanKHR {
    XrStructureType    type;
    void*              next;
    VkImage            image;
} XrSwapchainImageVulkanKHR;

typedef XrSwapchainImageVulkanKHR XrSwapchainImageVulkan2KHR;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `image` is a valid Vulkan `VkImage` to use.

If a given session was created with `XrGraphicsBindingVulkan2KHR`, the following conditions **must** apply.

- Calls to `xrEnumerateSwapchainImages` on an `XrSwapchain` in that session **must** return an array of `XrSwapchainImageVulkan2KHR` structures.
- Whenever an OpenXR function accepts an `XrSwapchainImageBaseHeader` pointer as a parameter in that session, the runtime **must** also accept a pointer to an `XrSwapchainImageVulkan2KHR`.

The OpenXR runtime **must** interpret the top-left corner of the swapchain image as the coordinate origin unless specified otherwise by extension functionality.

The OpenXR runtime **must** interpret the swapchain images in a clip space of positive Y pointing down, near Z plane at 0, and far Z plane at 1.

### Valid Usage (Implicit)

- The `XR_KHR_vulkan_enable2` extension **must** be enabled prior to using `XrSwapchainImageVulkan2KHR`
- **Note:** `XrSwapchainImageVulkan2KHR` is an alias for `XrSwapchainImageVulkanKHR`, so the following items replicate the implicit valid usage for `XrSwapchainImageVulkanKHR`
- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_VULKAN_KHR`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain. See also: `XrSwapchainImageFoveationVulkanFB`

### Swapchain Image Layout

When an application acquires a swapchain image by calling `xrAcquireSwapchainImage` in a session created using `XrGraphicsBindingVulkan2KHR`, the OpenXR runtime **must** guarantee that:

- The image has a memory layout compatible with `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` for color images, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` for depth images.
- The `VkQueue` specified in `XrGraphicsBindingVulkan2KHR` has ownership of the image.

When an application releases a swapchain image by calling `xrReleaseSwapchainImage`, in a session created using `XrGraphicsBindingVulkan2KHR`, the OpenXR runtime **must** interpret the image as:

- Having a memory layout compatible with `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` for color images, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` for depth images.
- Being owned by the `VkQueue` specified in `XrGraphicsBindingVulkan2KHR`.
- Being referenced by command buffers submitted to the `VkQueue` specified in `XrGraphicsBindingVulkan2KHR` which have not yet completed execution.

The application is responsible for transitioning the swapchain image back to the image layout and queue ownership that the OpenXR runtime requires. If the image is not in a layout compatible with the above specifications the runtime **may** exhibit undefined behavior.

### Swapchain Flag Bits

All `XrSwapchainUsageFlags` values passed in a session created using `XrGraphicsBindingVulkan2KHR` **must** be interpreted as follows by the runtime, so that the returned swapchain images used by the application may be used as if they were created with at least the specified `VkImageUsageFlagBits` or `VkImageCreateFlagBits` set.

| <b>XrSwapchainUsageFlagBits</b>   | <b>Corresponding Vulkan flag bit</b>                     |
|---|--|
| <code>XR_SWAPCHAIN_USAGE_COLOR_ATTACHMENT_BIT</code>  | <code>VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT</code>         |
| <code>XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT</code>  | <code>VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT</code> |
| <code>XR_SWAPCHAIN_USAGE_UNORDERED_ACCESS_BIT</code>  | <code>VK_IMAGE_USAGE_STORAGE_BIT</code>                  |
| <code>XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT</code>  | <code>VK_IMAGE_USAGE_TRANSFER_SRC_BIT</code>             |
| <code>XR_SWAPCHAIN_USAGE_TRANSFER_DST_BIT</code>  | <code>VK_IMAGE_USAGE_TRANSFER_DST_BIT</code>             |
| <code>XR_SWAPCHAIN_USAGE_SAMPLED_BIT</code>   | <code>VK_IMAGE_USAGE_SAMPLED_BIT</code>                  |
| <code>XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT</code>  | <code>VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT</code>          |
| <code>XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_KHR</code><br>(Added by <a href="#">XR_KHR_swapchain_usage_input_attachment_bit</a> and only available when that extension is enabled)                              | <code>VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT</code>         |
| <code>XR_SWAPCHAIN_USAGE_INPUT_ATTACHMENT_BIT_MND</code><br>(Added by the now deprecated <a href="#">XR_MND_swapchain_usage_input_attachment_bit</a> extension and only available when that extension is enabled) | <code>VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT</code>         |

## 12.21.5. Appendix

### Questions

- Should the [xrCreateVulkanDeviceKHR](#) and [xrCreateVulkanInstanceKHR](#) functions have an output parameter that returns the combined list of parameters used to create the Vulkan device/instance?
  - No. If the application is interested in capturing this data it can set the [pfnGetInstanceProcAddr](#) parameter to a local callback that captures the relevant information.

### Quick Reference

#### New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN2_KHR` (alias of `XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN_KHR`)
- `XR_TYPE_GRAPHICS_BINDING_VULKAN2_KHR` (alias of `XR_TYPE_GRAPHICS_BINDING_VULKAN_KHR`)
- `XR_TYPE_SWAPCHAIN_IMAGE_VULKAN2_KHR` (alias of `XR_TYPE_SWAPCHAIN_IMAGE_VULKAN_KHR`)

#### New Structures

- [XrVulkanInstanceCreateInfoKHR](#)
- [XrVulkanDeviceCreateInfoKHR](#)

- [XrVulkanGraphicsDeviceGetInfoKHR](#)
- [XrGraphicsBindingVulkan2KHR](#) (alias of [XrGraphicsBindingVulkanKHR](#))
- [XrSwapchainImageVulkan2KHR](#) (alias of [XrSwapchainImageVulkanKHR](#))
- [XrGraphicsRequirementsVulkan2KHR](#) (alias of [XrGraphicsRequirementsVulkanKHR](#))

#### New Functions

- [xrCreateVulkanInstanceKHR](#)
- [xrCreateVulkanDeviceKHR](#)
- [xrGetVulkanGraphicsDevice2KHR](#)
- [xrGetVulkanGraphicsRequirements2KHR](#)

#### Version History

- Revision 1, 2020-05-04 (Andres Rodriguez)
  - Initial draft
- Revision 2, 2021-01-21 (Rylie Pavlik, Collabora, Ltd.)
  - Document mapping for [XrSwapchainUsageFlags](#)

## 12.22. XR\_KHR\_vulkan\_swapchain\_format\_list

### Name String

[XR\\_KHR\\_vulkan\\_swapchain\\_format\\_list](#)

### Extension Type

Instance extension

### Registered Extension Number

15

### Revision

4

### Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_KHR\\_vulkan\\_enable](#)

### Last Modified Date

2020-01-01

## IP Status

No known IP claims.

## Contributors

Paul Pedriana, Oculus

Dan Ginsburg, Valve

## Overview

Vulkan has the `VK_KHR_image_format_list` extension which allows applications to tell the `vkCreateImage` function which formats the application intends to use when `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` is specified. This OpenXR extension exposes that Vulkan extension to OpenXR applications. In the same way that a Vulkan-based application can pass a `VkImageFormatListCreateInfo` struct to the `vkCreateImage` function, an OpenXR application can pass an identically configured `XrVulkanSwapchainFormatListCreateInfoKHR` structure to `xrCreateSwapchain`.

Applications using this extension to specify more than one swapchain format must create OpenXR swapchains with the `XR_SWAPCHAIN_USAGE_MUTABLE_FORMAT_BIT` bit set.

Runtimes implementing this extension **must** support the `XR_KHR_vulkan_enable` or the `XR_KHR_vulkan_enable2` extension. When `XR_KHR_vulkan_enable` is used, the runtime **must** add `VK_KHR_image_format_list` to the list of extensions enabled in `xrCreateVulkanDeviceKHR`.

## New Object Types

### New Flag Types

### New Enum Constants

`XrStructureType` enumeration is extended with:

```
XR_TYPE_VULKAN_SWAPCHAIN_FORMAT_LIST_CREATE_INFO_KHR
```

### New Enums

### New Structures

```
// Provided by XR_KHR_vulkan_swapchain_format_list
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
    XrStructureType    type;
    const void*        next;
    uint32_t           viewFormatCount;
    const VkFormat*    viewFormats;
} XrVulkanSwapchainFormatListCreateInfoKHR;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewFormatCount` is the number of view formats passed in `viewFormats`.
- `viewFormats` is an array of `VkFormat`.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_vulkan\\_swapchain\\_format\\_list](#) extension **must** be enabled prior to using [XrVulkanSwapchainFormatListCreateInfoKHR](#)
- `type` **must** be `XR_TYPE_VULKAN_SWAPCHAIN_FORMAT_LIST_CREATE_INFO_KHR`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `viewFormatCount` is not 0, `viewFormats` **must** be a pointer to an array of `viewFormatCount` valid `VkFormat` values

## New Functions

## Issues

## Version History

- Revision 1, 2017-09-13 (Paul Pedriana)
  - Initial proposal.
- Revision 2, 2018-06-21 (Bryce Hutchings)
  - Update reference of `XR_KHR_vulkan_extension_requirements` to `XR_KHR_vulkan_enable`
- Revision 3, 2020-01-01 (Andres Rodriguez)
  - Update for `XR_KHR_vulkan_enable2`
- Revision 4, 2021-01-21 (Rylie Pavlik, Collabora, Ltd.)

- Fix reference to the mutable-format bit in Vulkan.

## 12.23.

# XR\_KHR\_win32\_convert\_performance\_counter\_time

### Name String

`XR_KHR_win32_convert_performance_counter_time`

### Extension Type

Instance extension

### Registered Extension Number

36

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-01-24

### IP Status

No known IP claims.

### Contributors

Paul Pedriana, Oculus

Bryce Hutchings, Microsoft

### Overview

This extension provides two functions for converting between the Windows performance counter (QPC) time stamps and [XrTime](#). The [xrConvertWin32PerformanceCounterToTimeKHR](#) function converts from Windows performance counter time stamps to [XrTime](#), while the [xrConvertTimeToWin32PerformanceCounterKHR](#) function converts [XrTime](#) to Windows performance counter time stamps. The primary use case for this functionality is to be able to synchronize events between the local system and the OpenXR system.

### New Object Types

### New Flag Types

### New Enum Constants



## New Enums

## New Structures

## New Functions

To convert from a Windows performance counter time stamp to [XrTime](#), call:

```
// Provided by XR_KHR_win32_convert_performance_counter_time
XrResult xrConvertWin32PerformanceCounterToTimeKHR(
    XrInstance          instance,
    const LARGE_INTEGER* performanceCounter,
    XrTime*             time);
```

### Parameter Descriptions

- `instance` is an [XrInstance](#) handle previously created with [xrCreateInstance](#).
- `performanceCounter` is a time returned by [QueryPerformanceCounter](#).
- `time` is the resulting [XrTime](#) that is equivalent to the `performanceCounter`.

The [xrConvertWin32PerformanceCounterToTimeKHR](#) function converts a time stamp obtained by the [QueryPerformanceCounter](#) Windows function to the equivalent [XrTime](#).

If the output `time` cannot represent the input `performanceCounter`, the runtime **must** return `XR_ERROR_TIME_INVALID`.

### Valid Usage (Implicit)

- The [XR\\_KHR\\_win32\\_convert\\_performance\\_counter\\_time](#) extension **must** be enabled prior to calling [xrConvertWin32PerformanceCounterToTimeKHR](#)
- `instance` **must** be a valid [XrInstance](#) handle
- `performanceCounter` **must** be a pointer to a valid `LARGE_INTEGER` value
- `time` **must** be a pointer to an [XrTime](#) value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_TIME_INVALID`

To convert from `XrTime` to a Windows performance counter time stamp, call:

```
// Provided by XR_KHR_win32_convert_performance_counter_time
XrResult xrConvertTimeToWin32PerformanceCounterKHR(
    XrInstance          instance,
    XrTime              time,
    LARGE_INTEGER*     performanceCounter);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle previously created with `xrCreateInstance`.
- `time` is an `XrTime`.
- `performanceCounter` is the resulting Windows performance counter time stamp that is equivalent to the `time`.

The `xrConvertTimeToWin32PerformanceCounterKHR` function converts an `XrTime` to time as if generated by the `QueryPerformanceCounter` Windows function.

If the output `performanceCounter` cannot represent the input `time`, the runtime **must** return `XR_ERROR_TIME_INVALID`.

## Valid Usage (Implicit)

- The `XR_KHR_win32_convert_performance_counter_time` extension **must** be enabled prior to calling `xrConvertTimeToWin32PerformanceCounterKHR`
- `instance` **must** be a valid `XrInstance` handle
- `performanceCounter` **must** be a pointer to a `LARGE_INTEGER` value

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_TIME_INVALID`

## Issues

## Version History

- Revision 1, 2019-01-24 (Paul Pedriana)
  - Initial draft

## 12.24. XR\_EXT\_active\_action\_set\_priority

### Name String

`XR_EXT_active_action_set_priority`

### Extension Type

Instance extension

### Registered Extension Number

374

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-08-19

## IP Status

No known IP claims.

## Contributors

Jules Blok, Epic Games

Lachlan Ford, Microsoft

## Overview

The properties of an [XrActionSet](#) become immutable after it has been attached to a session. This currently includes the priority of the action set preventing the application from changing the priority number for the duration of the session.

Given that most runtimes do not actually require this number to be immutable this extension adds the ability to provide a different priority number for every [XrActiveActionSet](#) provided to [xrSyncActions](#).

When updating the action state with [xrSyncActions](#), the application **can** provide a pointer to an [XrActiveActionSetPrioritiesEXT](#) structure in the **next** chain of [XrActionsSyncInfo](#). This structure contains an array of [XrActiveActionSetPriorityEXT](#) structures mapping active action sets to their priority numbers.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_ACTIVE\\_ACTION\\_SET\\_PRIORITIES\\_EXT](#)

## New Enums

## New Structures

The [XrActiveActionSetPrioritiesEXT](#) structure is defined as:

```
// Provided by XR_EXT_active_action_set_priority
typedef struct XrActiveActionSetPrioritiesEXT {
    XrStructureType                type;
    const void*                    next;
    uint32_t                       actionSetPriorityCount;
    const XrActiveActionSetPriorityEXT* actionSetPriorities;
} XrActiveActionSetPrioritiesEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `actionSetPriorityCount` is an integer specifying the number of valid elements in the `actionSetPriorities` array.
- `actionSetPriorities` is a pointer to an array that maps action sets to their active priority numbers. If an action set is specified multiple times, the runtime **may** return `XR_ERROR_VALIDATION_FAILURE` from [xrSyncActions](#).

## Valid Usage (Implicit)

- The `XR_EXT_active_action_set_priority` extension **must** be enabled prior to using [XrActiveActionSetPrioritiesEXT](#)
- `type` **must** be `XR_TYPE_ACTIVE_ACTION_SET_PRIORITIES_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `actionSetPriorities` **must** be a pointer to an array of `actionSetPriorityCount` valid [XrActiveActionSetPriorityEXT](#) structures
- The `actionSetPriorityCount` parameter **must** be greater than `0`

The runtime **must** ignore any priority numbers for action sets that were not specified as an active action set in the [XrActionsSyncInfo](#) structure as this would have no effect.

The priority numbers provided in [XrActiveActionSetPriorityEXT](#) **must** override the priority number of the active action set starting with the [xrSyncActions](#) call it is provided to, until the first subsequent call to [xrSyncActions](#).

When a subsequent call is made to [xrSyncActions](#) where an active action set does not have a corresponding priority number specified in the [XrActiveActionSetPriorityEXT](#) structure the priority number for that action set **must** revert back to the priority number provided in [XrActionSetCreateInfo](#)

when that action set was created.

The `XrActiveActionSetPriorityEXT` structure is defined as:

```
// Provided by XR_EXT_active_action_set_priority
typedef struct XrActiveActionSetPriorityEXT {
    XrActionSet    actionSet;
    uint32_t       priorityOverride;
} XrActiveActionSetPriorityEXT;
```

### Member Descriptions

- `actionSet` is the handle of the `XrActionSet` to set the priority number for.
- `priorityOverride` is an integer specifying the priority of the action set while it is active.

### Valid Usage (Implicit)

- The `XR_EXT_active_action_set_priority` extension **must** be enabled prior to using `XrActiveActionSetPriorityEXT`
- `actionSet` **must** be a valid `XrActionSet` handle

### New Functions

### Issues

- Can the same action set have a different priority on each subaction path?
  - No. To avoid additional complexity each action set can only be specified once in the array of priorities which does not include the subaction path.

### Version History

- Revision 1, 2022-08-19 (Jules Blok)
  - Initial proposal.

## 12.25. XR\_EXT\_conformance\_automation

### Name String

`XR_EXT_conformance_automation`

## Extension Type

Instance extension

## Registered Extension Number

48

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-04-14

## IP Status

No known IP claims.

## Contributors

Lachlan Ford, Microsoft  
Rylie Pavlik, Collabora

## Overview

The XR\_EXT\_conformance\_automation allows conformance test and runtime developers to provide hints to the underlying runtime as to what input the test is expecting. This enables runtime authors to automate the testing of their runtime conformance. This is useful for achieving rapidly iterative runtime development whilst maintaining conformance for runtime releases.

This extension provides the following capabilities:

- The ability to toggle the active state of an input device.
- The ability to set the state of an input device button or other input component.
- The ability to set the location of the input device.

Applications **may** call these functions at any time. The runtime **must** do its best to honor the request of applications calling these functions, however it does not guarantee that any state change will be reflected immediately, at all, or with the exact value that was requested. Applications are thus advised to wait for the state change to be observable and to not assume that the value they requested will be the value observed. If any of the functions of this extension are called, control over input **must** be removed from the physical hardware of the system.

## Warning

This extension is **not** intended for use by non-conformance-test applications. A runtime **may** require a runtime-specified configuration such as a "developer mode" to be enabled before reporting support for this extension or providing a non-stub implementation of it.

**Do not** use this functionality in a non-conformance-test application!

### New Object Types

### New Flag Types

### New Enum Constants

### New Enums

### New Structures

### New Functions

```
// Provided by XR_EXT_conformance_automation
XrResult xrSetInputDeviceActiveEXT(
    XrSession          session,
    XrPath             interactionProfile,
    XrPath             topLevelPath,
    XrBool32           isActive);
```

## Parameter Descriptions

- **session** is the [XrSession](#) to set the input device state in.
- **interactionProfile** is the path representing the interaction profile of the input device (e.g. */interaction\_profiles/khr/simple\_controller*).
- **topLevelPath** is the path representing the input device (e.g. */user/hand/left*).
- **isActive** is the requested activation state of the input device.



## Valid Usage

- `session` **must** be a valid session handle.
- `topLevelPath` **must** be a valid top level path.

## Valid Usage (Implicit)

- The `XR_EXT_conformance_automation` extension **must** be enabled prior to calling `xrSetInputDeviceActiveEXT`
- `session` **must** be a valid `XrSession` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`

```
// Provided by XR_EXT_conformance_automation
XrResult xrSetInputDeviceStateBoolEXT(
    XrSession          session,
    XrPath             topLevelPath,
    XrPath             inputSourcePath,
    XrBool32           state);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to set the input device state in.
- `topLevelPath` is the path representing the input device (e.g. `/user/hand/left`).
- `inputSourcePath` is the full path of the input component for which we wish to set the state for (e.g. `/user/hand/left/input/select/click`).
- `state` is the requested boolean state of the input device.

## Valid Usage

- `session` **must** be a valid session handle.
- `topLevelPath` **must** be a valid top level path.
- `inputSourcePath` **must** be a valid input source path.

## Valid Usage (Implicit)

- The [XR\\_EXT\\_conformance\\_automation](#) extension **must** be enabled prior to calling [xrSetInputDeviceStateBoolEXT](#)
- `session` **must** be a valid [XrSession](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`

```
// Provided by XR_EXT_conformance_automation
XrResult xrSetInputDeviceStateFloatEXT(
    XrSession          session,
    XrPath             topLevelPath,
    XrPath             inputSourcePath,
    float              state);
```

## Parameter Descriptions

- **session** is the [XrSession](#) to set the input device state in.
- **topLevelPath** is the path representing the input device (e.g. */user/hand/left*).
- **inputSourcePath** is the full path of the input component for which we wish to set the state for (e.g. */user/hand/left/input/trigger/value*).
- **state** is the requested float state of the input device.

## Valid Usage

- **session** **must** be a valid session handle.
- **topLevelPath** **must** be a valid top level path.
- **inputSourcePath** **must** be a valid input source path.

## Valid Usage (Implicit)

- The [XR\\_EXT\\_conformance\\_automation](#) extension **must** be enabled prior to calling [xrSetInputDeviceStateFloatEXT](#)
- **session** **must** be a valid [XrSession](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`

```
// Provided by XR_EXT_conformance_automation
XrResult xrSetInputDeviceStateVector2fEXT(
    XrSession          session,
    XrPath             topLevelPath,
    XrPath             inputSourcePath,
    XrVector2f         state);
```

## Parameter Descriptions

- `session` is the `XrSession` to set the input device state in.
- `topLevelPath` is the path representing the input device (e.g. `/user/hand/left`).
- `inputSourcePath` is the full path of the input component for which we wish to set the state for (e.g. `/user/hand/left/input/thumbstick`).
- `state` is the requested two-dimensional state of the input device.

## Valid Usage

- `session` **must** be a valid session handle.
- `topLevelPath` **must** be a valid top level path.
- `inputSourcePath` **must** be a valid input source path.

## Valid Usage (Implicit)

- The `XR_EXT_conformance_automation` extension **must** be enabled prior to calling `xrSetInputDeviceStateVector2fEXT`
- `session` **must** be a valid `XrSession` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`

```
// Provided by XR_EXT_conformance_automation
XrResult xrSetInputDeviceLocationEXT(
    XrSession          session,
    XrPath             topLevelPath,
    XrPath             inputSourcePath,
    XrSpace            space,
    XrPosef            pose);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to set the input device state in.
- `topLevelPath` is the path representing the input device (e.g. `/user/hand/left`).
- `inputSourcePath` is the full path of the input component for which we wish to set the pose for (e.g. `/user/hand/left/input/grip/pose`).
- `pose` is the requested pose state of the input device.

## Valid Usage

- `session` **must** be a valid session handle.
- `topLevelPath` **must** be a valid top level path.
- `inputSourcePath` **must** be a valid input source path.
- `space` **must** be a valid [XrSpace](#).
- `pose` **must** be a valid [XrPosef](#).

## Valid Usage (Implicit)

- The [XR\\_EXT\\_conformance\\_automation](#) extension **must** be enabled prior to calling [xrSetInputDeviceLocationEXT](#)
- `session` **must** be a valid [XrSession](#) handle
- `space` **must** be a valid [XrSpace](#) handle
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`

### New Function Pointers

### Issues

None

### Version History

- Revision 1, 2019-10-01 (Lachlan Ford)
  - Initial draft
- Revision 2, 2021-03-04 (Rylie Pavlik)
  - Correct errors in function parameter documentation.
- Revision 3, 2021-04-14 (Rylie Pavlik)
  - Fix missing error code

## 12.26. XR\_EXT\_debug\_utils

### Name String

`XR_EXT_debug_utils`

### Extension Type

Instance extension

## Registered Extension Number

20

## Revision

5

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-04-14

## IP Status

No known IP claims.

## Contributors

Mark Young, LunarG  
Karl Schultz, LunarG  
Rylie Pavlik, Collabora

## Overview

Due to the nature of the OpenXR interface, there is very little error information available to the developer and application. By using the [XR\\_EXT\\_debug\\_utils](#) extension, developers **can** obtain more information. When combined with validation layers, even more detailed feedback on the application's use of OpenXR will be provided.

This extension provides the following capabilities:

- The ability to create a debug messenger which will pass along debug messages to an application supplied callback.
- The ability to identify specific OpenXR handles using a name to improve tracking.

### 12.26.1. Object Debug Annotation

It can be useful for an application to provide its own content relative to a specific OpenXR handle.

#### Object Naming

[xrSetDebugUtilsObjectNameEXT](#) allows application developers to associate user-defined information with OpenXR handles.

This is useful when paired with the callback that you register when creating an [XrDebugUtilsMessengerEXT](#) object. When properly used, debug messages will contain not only the corresponding object handle, but the associated object name as well.



An application can change the name associated with an object simply by calling `xrSetDebugUtilsObjectNameEXT` again with a new string. If the `objectName` member of the `XrDebugUtilsObjectNameInfoEXT` structure is an empty string, then any previously set name is removed.

### 12.26.2. Debug Messengers

OpenXR allows an application to register arbitrary number of callbacks with all the OpenXR components wishing to report debug information. Some callbacks **can** log the information to a file, others **can** cause a debug break point or any other behavior defined by the application. A primary producer of callback messages are the validation layers. If the extension is enabled, an application **can** register callbacks even when no validation layers are enabled. The OpenXR loader, other layers, and runtimes **may** also produce callback messages.

The debug messenger will provide detailed feedback on the application's use of OpenXR when events of interest occur. When an event of interest does occur, the debug messenger will submit a debug message to the debug callback that was provided during its creation. Additionally, the debug messenger is responsible with filtering out debug messages that the callback isn't interested in and will only provide desired debug messages.

### 12.26.3. Debug Message Categorization

Messages that are triggered by the debug messenger are categorized by their message type and severity. Additionally, each message has a string value identifying its `messageId`. These 3 bits of information can be used to filter out messages so you only receive reports on the messages you desire. In fact, during debug messenger creation, the severity and type flag values are provided to indicate what messages should be allowed to trigger the user's callback.

#### Message Type

The message type indicates the general category the message falls under. Currently we have the following message types:

Table 4. `XR_EXT_debug_utils` Message Type Flag Descriptions

| Enum   | Description   |
|--|---|
| <code>XR_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT</code>     | Specifies a general purpose event type. This is typically a non-validation, non-performance event.                            |
| <code>XR_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT</code>  | Specifies an event caused during a validation against the OpenXR specification that <b>may</b> indicate invalid OpenXR usage. |
| <code>XR_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT</code> | Specifies a potentially non-optimal use of OpenXR.  |

| Enum   | Description   |
|--|---|
| <code>XR_DEBUG_UTILS_MESSAGE_TYPE_CONFORMANCE_BIT_EXT</code> | Specifies a non-conformant OpenXR result. This is typically caused by a layer or runtime returning non-conformant data. |

A message may correspond to more than one type. For example, if a validation warning also could impact performance, then the message might be identified with both the `XR_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT` and `XR_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT` flag bits.

## Message Severity

The severity of a message is a flag that indicates how important the message is using standard logging naming. The severity flag bit values are shown in the following table.

Table 5. `XR_EXT_debug_utils` Message Severity Flag Descriptions

| Enum   | Description  |
|--|--|
| <code>XR_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT</code> | Specifies the most verbose output indicating all diagnostic messages from the OpenXR loader, layers, and drivers should be captured.   |
| <code>XR_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT</code>    | Specifies an informational message such as resource details that might be handy when debugging an application.   |
| <code>XR_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT</code> | Specifies use of OpenXR that could be an application bug. Such cases <b>may</b> not be immediately harmful, such as providing too many swapchain images. Other cases <b>may</b> point to behavior that is almost certainly bad when unintended, such as using a swapchain image whose memory has not been filled. In general, if you see a warning but you know that the behavior is intended/desired, then simply ignore the warning. |
| <code>XR_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT</code>   | Specifies an error that <b>may</b> cause undefined behavior, including an application crash.   |

### Note



The values of `XrDebugUtilsMessageSeverityFlagBitsEXT` are sorted based on severity. The higher the flag value, the more severe the message. This allows for simple boolean operation comparisons when looking at `XrDebugUtilsMessageSeverityFlagBitsEXT` values.

## Message IDs

The [XrDebugUtilsMessengerCallbackDataEXT](#) structure contains a `messageId` that may be a string identifying the message ID for the triggering debug message. This may be blank, or it may simply contain the name of an OpenXR component (like "OpenXR Loader"). However, when certain API layers or runtimes are used, especially the OpenXR `core_validation` API layer, then this value is intended to uniquely identify the message generated. If a certain warning/error message constantly fires, a user can simply look at the unique ID in their callback handler and manually filter it out.

For validation layers, this `messageId` value actually can be used to find the section of the OpenXR specification that the layer believes to have been violated. See the `core_validation` API Layer documentation for more information on how this can be done.

### 12.26.4. Session Labels

All OpenXR work is performed inside of an [XrSession](#). There are times that it helps to label areas in your OpenXR session to allow easier debugging. This can be especially true if your application creates more than one session. There are two kinds of labels provided in this extension:

- Region labels
- Individual labels

To begin identifying a region using a debug label inside a session, you may use the [xrSessionBeginDebugUtilsLabelRegionEXT](#) function. Calls to [xrSessionBeginDebugUtilsLabelRegionEXT](#) may be nested allowing you to identify smaller and smaller labeled regions within your code. Using this, you can build a "call-stack" of sorts with labels since any logging callback will contain the list of all active session label regions.

To end the last session label region that was begun, you **must** call [xrSessionEndDebugUtilsLabelRegionEXT](#). Each [xrSessionBeginDebugUtilsLabelRegionEXT](#) **must** have a matching [xrSessionEndDebugUtilsLabelRegionEXT](#). All of a session's label regions **must** be closed before the [xrDestroySession](#) function is called for the given [XrSession](#).

An individual debug label may be inserted at any time using [xrSessionInsertDebugUtilsLabelEXT](#). The [xrSessionInsertDebugUtilsLabelEXT](#) is used to indicate a particular location within the execution of the application's session functions. The next call to [xrSessionInsertDebugUtilsLabelEXT](#), [xrSessionBeginDebugUtilsLabelRegionEXT](#), or [xrSessionEndDebugUtilsLabelRegionEXT](#) overrides this value.

## New Object Types

```
XR_DEFINE_HANDLE(XrDebugUtilsMessengerEXT)
```

[XrDebugUtilsMessengerEXT](#) represents a callback function and associated filters registered with the runtime.

## New Flag Types

```
typedef XrFlags64 XrDebugUtilsMessageSeverityFlagsEXT;
```

```
// Flag bits for XrDebugUtilsMessageSeverityFlagsEXT
static const XrDebugUtilsMessageSeverityFlagsEXT
XR_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT = 0x00000001;
static const XrDebugUtilsMessageSeverityFlagsEXT
XR_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT = 0x00000010;
static const XrDebugUtilsMessageSeverityFlagsEXT
XR_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT = 0x00000100;
static const XrDebugUtilsMessageSeverityFlagsEXT
XR_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT = 0x00001000;
```

```
typedef XrFlags64 XrDebugUtilsMessageTypeFlagsEXT;
```

```
// Flag bits for XrDebugUtilsMessageTypeFlagsEXT
static const XrDebugUtilsMessageTypeFlagsEXT XR_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT
= 0x00000001;
static const XrDebugUtilsMessageTypeFlagsEXT
XR_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT = 0x00000002;
static const XrDebugUtilsMessageTypeFlagsEXT
XR_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT = 0x00000004;
static const XrDebugUtilsMessageTypeFlagsEXT
XR_DEBUG_UTILS_MESSAGE_TYPE_CONFORMANCE_BIT_EXT = 0x00000008;
```

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT`
- `XR_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT`

- `XR_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT`
- `XR_TYPE_DEBUG_UTILS_LABEL_EXT`

## New Enums

## New Structures

```
// Provided by XR_EXT_debug_utils
typedef struct XrDebugUtilsObjectNameInfoEXT {
    XrStructureType    type;
    const void*        next;
    XrObjectType        objectType;
    uint64_t           objectHandle;
    const char*        objectName;
} XrDebugUtilsObjectNameInfoEXT;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `objectType` is an `XrObjectType` specifying the type of the object to be named.
- `objectHandle` is the object to be named.
- `objectName` is a `NULL` terminated UTF-8 string specifying the name to apply to `objectHandle`.

### Valid Usage

- If `objectType` is `XR_OBJECT_TYPE_UNKNOWN`, `objectHandle` **must** not be `XR_NULL_HANDLE`
- If `objectType` is not `XR_OBJECT_TYPE_UNKNOWN`, `objectHandle` **must** be `XR_NULL_HANDLE` or an OpenXR handle of the type associated with `objectType`

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to using `XrDebugUtilsObjectNameInfoEXT`
- `type` **must** be `XR_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `objectType` **must** be a valid `XrObjectType` value
- If `objectName` is not `NULL`, `objectName` **must** be a null-terminated UTF-8 string

```
// Provided by XR_EXT_debug_utils
typedef struct XrDebugUtilsLabelEXT {
    XrStructureType    type;
    const void*        next;
    const char*        labelName;
} XrDebugUtilsLabelEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `labelName` is a `NULL` terminated UTF-8 string specifying the label name.

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to using `XrDebugUtilsLabelEXT`
- `type` **must** be `XR_TYPE_DEBUG_UTILS_LABEL_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `labelName` **must** be a null-terminated UTF-8 string

```
// Provided by XR_EXT_debug_utils
typedef struct XrDebugUtilsMessengerCallbackDataEXT {
    XrStructureType          type;
    const void*              next;
    const char*              messageId;
    const char*              functionName;
    const char*              message;
    uint32_t                 objectCount;
    XrDebugUtilsObjectNameInfoEXT* objects;
    uint32_t                 sessionLabelCount;
    XrDebugUtilsLabelEXT*   sessionLabels;
} XrDebugUtilsMessengerCallbackDataEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `messageId` is a `NULL` terminated string that identifies the message in a unique way. If the callback is triggered by a validation layer, this string corresponds the Valid Usage ID (VUID) that can be used to jump to the appropriate location in the OpenXR specification. This value **may** be `NULL` if no unique message identifier is associated with the message.
- `functionName` is a `NULL` terminated string that identifies the OpenXR function that was executing at the time the message callback was triggered. This value **may** be `NULL` in cases where it is difficult to determine the originating OpenXR function.
- `message` is a `NULL` terminated string detailing the trigger conditions.
- `objectCount` is a count of items contained in the `objects` array. This may be `0`.
- `objects` is `NULL` or a pointer to an array of [XrDebugUtilsObjectNameInfoEXT](#) objects related to the detected issue. The array is roughly in order of importance, but the 0th element is always guaranteed to be the most important object for this message.
- `sessionLabelCount` is a count of items contained in the `sessionLabels` array. This may be `0`.
- `sessionLabels` is `NULL` or a pointer to an array of [XrDebugUtilsLabelEXT](#) active in the current [XrSession](#) at the time the callback was triggered. Refer to [Session Labels](#) for more information.

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to using `XrDebugUtilsMessengerCallbackDataEXT`
- `type` **must** be `XR_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `messageId` is not `NULL`, `messageId` **must** be a null-terminated UTF-8 string
- If `functionName` is not `NULL`, `functionName` **must** be a null-terminated UTF-8 string
- `message` **must** be a null-terminated UTF-8 string

An `XrDebugUtilsMessengerCallbackDataEXT` is a messenger object that handles passing along debug messages to a provided debug callback.



### Note

This structure should only be considered valid during the lifetime of the triggered callback.

The labels listed inside `sessionLabels` are organized in time order, with the most recently generated label appearing first, and the oldest label appearing last.

```
// Provided by XR_EXT_debug_utils
typedef struct XrDebugUtilsMessengerCreateInfoEXT {
    XrStructureType          type;
    const void*              next;
    XrDebugUtilsMessageSeverityFlagsEXT messageSeverities;
    XrDebugUtilsMessageTypeFlagsEXT messageTypes;
    PFN_xrDebugUtilsMessengerCallbackEXT userCallback;
    void*                     userData;
} XrDebugUtilsMessengerCreateInfoEXT;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `messageSeverities` is a bitmask of [XrDebugUtilsMessageSeverityFlagBitsEXT](#) specifying which severity of event(s) that will cause this callback to be called.
- `messageTypes` is a combination of [XrDebugUtilsMessageTypeFlagBitsEXT](#) specifying which type of event(s) will cause this callback to be called.
- `userCallback` is the application defined callback function to call.
- `userData` is arbitrary user data to be passed to the callback.

## Valid Usage

- `userCallback` **must** be a valid [PFN\\_xrDebugUtilsMessengerCallbackEXT](#)

## Valid Usage (Implicit)

- The [XR\\_EXT\\_debug\\_utils](#) extension **must** be enabled prior to using [XrDebugUtilsMessengerCreateInfoEXT](#)
- `type` **must** be `XR_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `messageSeverities` **must** be a valid combination of [XrDebugUtilsMessageSeverityFlagBitsEXT](#) values
- `messageSeverities` **must** not be `0`
- `messageTypes` **must** be a valid combination of [XrDebugUtilsMessageTypeFlagBitsEXT](#) values
- `messageTypes` **must** not be `0`
- `userCallback` **must** be a valid [PFN\\_xrDebugUtilsMessengerCallbackEXT](#) value

For each [XrDebugUtilsMessengerEXT](#) that is created the [XrDebugUtilsMessengerCreateInfoEXT::messageSeverities](#) and [XrDebugUtilsMessengerCreateInfoEXT::messageTypes](#) determine when that [XrDebugUtilsMessengerCreateInfoEXT::userCallback](#) is called. The process to determine if the user's `userCallback` is triggered when an event occurs is as follows:

- The runtime will perform a bitwise AND of the event's [XrDebugUtilsMessageSeverityFlagBitsEXT](#) with the [XrDebugUtilsMessengerCreateInfoEXT::messageSeverities](#) provided during creation of the [XrDebugUtilsMessengerEXT](#) object.

- If this results in 0, the message is skipped.
- The runtime will perform bitwise AND of the event's [XrDebugUtilsMessageTypeFlagBitsEXT](#) with the [XrDebugUtilsMessengerCreateInfoEXT::messageTypes](#) provided during the creation of the [XrDebugUtilsMessengerEXT](#) object.
- If this results in 0, the message is skipped.
- If the message of the current event is not skipped, the callback will be called with the message.

The callback will come directly from the component that detected the event, unless some other layer intercepts the calls for its own purposes (filter them in a different way, log to a system error log, etc.).

An application **can** receive multiple callbacks if multiple [XrDebugUtilsMessengerEXT](#) objects are created. A callback will always be executed in the same thread as the originating OpenXR call.



*Note*

A callback **can** be called from multiple threads simultaneously if the application is making OpenXR calls from multiple threads.

## New Functions

```
// Provided by XR_EXT_debug_utils
XrResult xrSetDebugUtilsObjectNameEXT(
    XrInstance instance,
    const XrDebugUtilsObjectNameInfoEXT* nameInfo);
```

### Parameter Descriptions

- **instance** is the [XrInstance](#) that the object was created under.
- **nameInfo** is a pointer to an instance of the [XrDebugUtilsObjectNameInfoEXT](#) structure specifying the parameters of the name to set on the object.

### Valid Usage

- In the structure pointed to by **nameInfo**, [XrDebugUtilsObjectNameInfoEXT::objectType](#) **must** not be [XR\\_OBJECT\\_TYPE\\_UNKNOWN](#)
- In the structure pointed to by **nameInfo**, [XrDebugUtilsObjectNameInfoEXT::objectHandle](#) **must** not be [XR\\_NULL\\_HANDLE](#)

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to calling `xrSetDebugUtilsObjectNameEXT`
- `instance` **must** be a valid `XrInstance` handle
- `nameInfo` **must** be a pointer to a valid `XrDebugUtilsObjectNameInfoEXT` structure

## Thread Safety

- Access to the `objectHandle` member of the `nameInfo` parameter **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

Applications **may** change the name associated with an object simply by calling `xrSetDebugUtilsObjectNameEXT` again with a new string. If `XrDebugUtilsObjectNameInfoEXT::objectName` is an empty string, then any previously set name is removed.

```
// Provided by XR_EXT_debug_utils
XrResult xrCreateDebugUtilsMessengerEXT(
    XrInstance                instance,
    const XrDebugUtilsMessengerCreateInfoEXT* createInfo,
    XrDebugUtilsMessengerEXT* messenger);
```

## Parameter Descriptions

- `instance` is the instance the messenger will be used with.
- `createInfo` points to an `XrDebugUtilsMessengerCreateInfoEXT` structure, which contains the callback pointer as well as defines the conditions under which this messenger will trigger the callback.
- `messenger` is a pointer to which the created `XrDebugUtilsMessengerEXT` object is returned.

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to calling `xrCreateDebugUtilsMessengerEXT`
- `instance` **must** be a valid `XrInstance` handle
- `createInfo` **must** be a pointer to a valid `XrDebugUtilsMessengerCreateInfoEXT` structure
- `messenger` **must** be a pointer to an `XrDebugUtilsMessengerEXT` handle

## Thread Safety

- Access to `instance`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The application **must** ensure that `xrCreateDebugUtilsMessengerEXT` is not executed in parallel with any OpenXR function that is also called with `instance` or child of `instance`.

When an event of interest occurs a debug messenger calls its [XrDebugUtilsMessengerCreateInfoEXT::userCallback](#) with a debug message from the producer of the event. Additionally, the debug messenger **must** filter out any debug messages that the application's callback is not interested in based on [XrDebugUtilsMessengerCreateInfoEXT](#) flags, as described below.

```
// Provided by XR_EXT_debug_utils
XrResult xrDestroyDebugUtilsMessengerEXT(
    XrDebugUtilsMessengerEXT messenger);
```

### Parameter Descriptions

- **messenger** the [XrDebugUtilsMessengerEXT](#) object to destroy. **messenger** is an externally synchronized object and **must** not be used on more than one thread at a time. This means that [xrDestroyDebugUtilsMessengerEXT](#) **must** not be called when a callback is active.

### Valid Usage (Implicit)

- The [XR\\_EXT\\_debug\\_utils](#) extension **must** be enabled prior to calling [xrDestroyDebugUtilsMessengerEXT](#)
- **messenger** **must** be a valid [XrDebugUtilsMessengerEXT](#) handle

### Thread Safety

- Access to **messenger** **must** be externally synchronized
- Access to the [XrInstance](#) used to create **messenger**, and all of its child handles **must** be externally synchronized

### Return Codes

#### Success

- [XR\\_SUCCESS](#)

#### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)

The application **must** ensure that `xrDestroyDebugUtilsMessengerEXT` is not executed in parallel with any OpenXR function that is also called with the `instance` or child of `instance` that it was created with.

```
// Provided by XR_EXT_debug_utils
XrResult          xrSubmitDebugUtilsMessageEXT(
    XrInstance     instance,
    XrDebugUtilsMessageSeverityFlagsEXT messageSeverity,
    XrDebugUtilsMessageTypeFlagsEXT messageTypes,
    const XrDebugUtilsMessengerCallbackDataEXT* callbackData);
```

### Parameter Descriptions

- `instance` is the debug stream's `XrInstance`.
- `messageSeverity` is a single bit value of `XrDebugUtilsMessageSeverityFlagsEXT` severity of this event/message.
- `messageTypes` is an `XrDebugUtilsMessageTypeFlagsEXT` bitmask of `XrDebugUtilsMessageTypeFlagBitsEXT` specifying which types of event to identify this message with.
- `callbackData` contains all the callback related data in the `XrDebugUtilsMessengerCallbackDataEXT` structure.

### Valid Usage

- For each structure in `XrDebugUtilsMessengerCallbackDataEXT::objects`, the value of `XrDebugUtilsObjectNameInfoEXT::objectType` **must** not be `XR_OBJECT_TYPE_UNKNOWN`

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to calling `xrSubmitDebugUtilsMessageEXT`
- `instance` **must** be a valid `XrInstance` handle
- `messageSeverity` **must** be a valid combination of `XrDebugUtilsMessageSeverityFlagBitsEXT` values
- `messageSeverity` **must** not be `0`
- `messageTypes` **must** be a valid combination of `XrDebugUtilsMessageTypeFlagBitsEXT` values
- `messageTypes` **must** not be `0`
- `callbackData` **must** be a pointer to a valid `XrDebugUtilsMessengerCallbackDataEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`

The application **can** also produce a debug message, and submit it into the OpenXR messaging system.

The call will propagate through the layers and generate callback(s) as indicated by the message's flags. The parameters are passed on to the callback in addition to the `userData` value that was defined at the time the messenger was created.

```
// Provided by XR_EXT_debug_utils
XrResult xrSessionBeginDebugUtilsLabelRegionEXT(
    XrSession session,
    const XrDebugUtilsLabelEXT* labelInfo);
```

## Parameter Descriptions

- `session` is the `XrSession` that a label region should be associated with.
- `labelInfo` is the `XrDebugUtilsLabelEXT` containing the label information for the region that should be begun.

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to calling `xrSessionBeginDebugUtilsLabelRegionEXT`
- `session` **must** be a valid `XrSession` handle
- `labelInfo` **must** be a pointer to a valid `XrDebugUtilsLabelEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrSessionBeginDebugUtilsLabelRegionEXT` function begins a label region within `session`.

```
// Provided by XR_EXT_debug_utils
XrResult xrSessionEndDebugUtilsLabelRegionEXT(
    XrSession session);
```



## Parameter Descriptions

- `session` is the [XrSession](#) that a label region should be associated with.

## Valid Usage

- [xrSessionEndDebugUtilsLabelRegionEXT](#) **must** be called only after a matching [xrSessionBeginDebugUtilsLabelRegionEXT](#).

## Valid Usage (Implicit)

- The [XR\\_EXT\\_debug\\_utils](#) extension **must** be enabled prior to calling [xrSessionEndDebugUtilsLabelRegionEXT](#)
- `session` **must** be a valid [XrSession](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

This function ends the last label region begun with the [xrSessionBeginDebugUtilsLabelRegionEXT](#) function within the same `session`.

```
// Provided by XR_EXT_debug_utils
XrResult xrSessionInsertDebugUtilsLabelEXT(
    XrSession session,
    const XrDebugUtilsLabelEXT* labelInfo);
```

## Parameter Descriptions

- `session` is the `XrSession` that a label region should be associated with.
- `labelInfo` is the `XrDebugUtilsLabelEXT` containing the label information for the region that should be begun.

## Valid Usage (Implicit)

- The `XR_EXT_debug_utils` extension **must** be enabled prior to calling `xrSessionInsertDebugUtilsLabelEXT`
- `session` **must** be a valid `XrSession` handle
- `labelInfo` **must** be a pointer to a valid `XrDebugUtilsLabelEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrSessionInsertDebugUtilsLabelEXT` function inserts an individual label within `session`. The individual labels are useful for different reasons based on the type of debugging scenario. When used with something active like a profiler or debugger, it identifies a single point of time. When used with logging, the individual label identifies that a particular location has been passed at the point the log message is triggered. Because of this usage, individual labels only exist in a log until the next call to any

of the label functions:

- [xrSessionBeginDebugUtilsLabelRegionEXT](#)
- [xrSessionEndDebugUtilsLabelRegionEXT](#)
- [xrSessionInsertDebugUtilsLabelEXT](#)

## New Function Pointers

```
// Provided by XR_EXT_debug_utils
typedef XrBool32 (XRAPI_PTR *PFN_xrDebugUtilsMessengerCallbackEXT)(
    XrDebugUtilsMessageSeverityFlagsEXT      messageSeverity,
    XrDebugUtilsMessageTypeFlagsEXT        messageTypes,
    const XrDebugUtilsMessengerCallbackDataEXT* callbackData,
    void*                                    userData);
```

### Parameter Descriptions

- `messageSeverity` indicates the single bit value of [XrDebugUtilsMessageSeverityFlagsEXT](#) that triggered this callback.
- `messageTypes` indicates the [XrDebugUtilsMessageTypeFlagsEXT](#) specifying which types of event triggered this callback.
- `callbackData` contains all the callback related data in the [XrDebugUtilsMessengerCallbackDataEXT](#) structure.
- `userData` is the user data provided when the [XrDebugUtilsMessengerEXT](#) was created.

The callback **must** not call [xrDestroyDebugUtilsMessengerEXT](#).

The callback returns an [XrBool32](#) that indicates to the calling layer the application's desire to abort the call. A value of `XR_TRUE` indicates that the application wants to abort this call. If the application returns `XR_FALSE`, the function **must** not be aborted. Applications **should** always return `XR_FALSE` so that they see the same behavior with and without validation layers enabled.

If the application returns `XR_TRUE` from its callback and the OpenXR call being aborted returns an [XrResult](#), the layer will return `XR_ERROR_VALIDATION_FAILURE`.

The object pointed to by `callbackData` (and any pointers in it recursively) **must** be valid during the lifetime of the triggered callback. It **may** become invalid afterwards.

## Examples

### Example 1

`XR_EXT_debug_utils` allows an application to register multiple callbacks with any OpenXR component wishing to report debug information. Some callbacks may log the information to a file, others may cause a debug break point or other application defined behavior. An application **can** register callbacks even when no validation layers are enabled, but they will only be called for loader and, if implemented, driver events.

To capture events that occur while creating or destroying an instance an application **can** link an `XrDebugUtilsMessengerCreateInfoEXT` structure to the next element of the `XrInstanceCreateInfo` structure given to `xrCreateInstance`. This callback is only valid for the duration of the `xrCreateInstance` and the `xrDestroyInstance` call. Use `xrCreateDebugUtilsMessengerEXT` to create persistent callback objects.

Example uses: Create three callback objects. One will log errors and warnings to the debug console using Windows `OutputDebugString`. The second will cause the debugger to break at that callback when an error happens and the third will log warnings to stdout.

```
extern XrInstance instance; // previously initialized

// Must call extension functions through a function pointer:
PFN_xrCreateDebugUtilsMessengerEXT pfnCreateDebugUtilsMessengerEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateDebugUtilsMessengerEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnCreateDebugUtilsMessengerEXT)));

PFN_xrDestroyDebugUtilsMessengerEXT pfnDestroyDebugUtilsMessengerEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrDestroyDebugUtilsMessengerEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnDestroyDebugUtilsMessengerEXT)));

XrDebugUtilsMessengerCreateInfoEXT callback1 = {
    XR_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT, // type
    NULL, // next
    XR_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT | // messageSeverities
        XR_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT,
    XR_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT | // messageTypes
        XR_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT,
    myOutputDebugString, // userCallback
    NULL // userData
};
XrDebugUtilsMessengerEXT messenger1 = XR_NULL_HANDLE;
CHK_XR(pfnCreateDebugUtilsMessengerEXT(instance, &callback1, &messenger1));

callback1.messageSeverities = XR_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
callback1.userCallback = myDebugBreak;
callback1.userData = NULL;
XrDebugUtilsMessengerEXT messenger2 = XR_NULL_HANDLE;
CHK_XR(pfnCreateDebugUtilsMessengerEXT(instance, &callback1, &messenger2));
```

```

XrDebugUtilsMessengerCreateInfoEXT callback3 = {
    XR_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT, // type
    NULL, // next
    XR_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT, // messageSeverities
    XR_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT | // messageTypes
        XR_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT,
    myStdOutLogger, // userCallback
    NULL // userData
};
XrDebugUtilsMessengerEXT messenger3 = XR_NULL_HANDLE;
CHK_XR(pfnCreateDebugUtilsMessengerEXT(instance, &callback3, &messenger3));

// ...

// Remove callbacks when cleaning up
pfnDestroyDebugUtilsMessengerEXT(messenger1);
pfnDestroyDebugUtilsMessengerEXT(messenger2);
pfnDestroyDebugUtilsMessengerEXT(messenger3);

```

## Example 2

Associate a name with an [XrSpace](#), for easier debugging in external tools or with validation layers that can print a friendly name when referring to objects in error messages.

```

extern XrInstance instance; // previously initialized
extern XrSpace space;      // previously initialized

// Must call extension functions through a function pointer:
PFN_xrSetDebugUtilsObjectNameEXT pfnSetDebugUtilsObjectNameEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrSetDebugUtilsObjectNameEXT",
                                reinterpret_cast<PFN_xrVoidFunction*>(
                                    &pfnSetDebugUtilsObjectNameEXT)));

// Set a name on the space
const XrDebugUtilsObjectNameInfoEXT spaceNameInfo = {
    XR_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT, // type
    NULL, // next
    XR_OBJECT_TYPE_SPACE, // objectType
    (uint64_t)space, // objectHandle
    "My Object-Specific Space", // objectName
};

pfnSetDebugUtilsObjectNameEXT(instance, &spaceNameInfo);

// A subsequent error might print:
// Space "My Object-Specific Space" (0xc0dec0dedeadbeef) is used
// with an XrSession that is not it's parent.

```

### Example 3

Labeling the workload with naming information so that any form of analysis can display a more usable visualization of where actions occur in the lifetime of a session.

```

extern XrInstance instance; // previously initialized
extern XrSession session;  // previously initialized

// Must call extension functions through a function pointer:

PFN_xrSessionBeginDebugUtilsLabelRegionEXT pfnSessionBeginDebugUtilsLabelRegionEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrSessionBeginDebugUtilsLabelRegionEXT",
                                reinterpret_cast<PFN_xrVoidFunction*>(
                                    &pfnSessionBeginDebugUtilsLabelRegionEXT)));

PFN_xrSessionEndDebugUtilsLabelRegionEXT pfnSessionEndDebugUtilsLabelRegionEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrSessionEndDebugUtilsLabelRegionEXT",
                                reinterpret_cast<PFN_xrVoidFunction*>(
                                    &pfnSessionEndDebugUtilsLabelRegionEXT)));

PFN_xrSessionInsertDebugUtilsLabelEXT pfnSessionInsertDebugUtilsLabelEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrSessionInsertDebugUtilsLabelEXT",

```

```

        reinterpret_cast<PFN_xrVoidFunction*>(
            &pfnSessionInsertDebugUtilsLabelEXT));

XrSessionBeginInfo session_begin_info = {
    XR_TYPE_SESSION_BEGIN_INFO,
    nullptr,
    XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO
};
xrBeginSession(session, &session_begin_info);

const XrDebugUtilsLabelEXT session_active_region_label = {
    XR_TYPE_DEBUG_UTILS_LABEL_EXT, // type
    NULL,                          // next
    "Session active",              // labelName
};

// Start an annotated region of calls under the 'Session Active' name
pfnSessionBeginDebugUtilsLabelRegionEXT(session, &session_active_region_label);

// Brackets added for clarity
{
    XrDebugUtilsLabelEXT individual_label = {
        XR_TYPE_DEBUG_UTILS_LABEL_EXT, // type
        NULL,                          // next
        "WaitFrame",                  // labelName
    };

    const char wait_frame_label[] = "WaitFrame";
    individual_label.labelName = wait_frame_label;
    pfnSessionInsertDebugUtilsLabelEXT(session, &individual_label);
    XrFrameWaitInfo wait_frame_info; // initialization omitted for readability
    XrFrameState frame_state = {XR_TYPE_FRAME_STATE, nullptr};
    xrWaitFrame(session, &wait_frame_info, &frame_state);

    // Do stuff 1

    const XrDebugUtilsLabelEXT session_frame_region_label = {
        XR_TYPE_DEBUG_UTILS_LABEL_EXT, // type
        NULL,                          // next
        "Session Frame 123",          // labelName
    };

    // Start an annotated region of calls under the 'Session Frame 123' name
    pfnSessionBeginDebugUtilsLabelRegionEXT(session, &session_frame_region_label);

    // Brackets added for clarity
    {

```

```

const char begin_frame_label[] = "BeginFrame";
individual_label.labelName = begin_frame_label;
pfnSessionInsertDebugUtilsLabelEXT(session, &individual_label);

XrFrameBeginInfo begin_frame_info; // initialization omitted for readability
xrBeginFrame(session, &begin_frame_info);

// Do stuff 2

const char end_frame_label[] = "EndFrame";
individual_label.labelName = end_frame_label;
pfnSessionInsertDebugUtilsLabelEXT(session, &individual_label);

XrFrameEndInfo end_frame_info; // initialization omitted for readability
xrEndFrame(session, &end_frame_info);
}

// End the session/begun region started above
// (in this case it's the "Session Frame 123" label)
pfnSessionEndDebugUtilsLabelRegionEXT(session);
}

// End the session/begun region started above
// (in this case it's the "Session Active" label)
pfnSessionEndDebugUtilsLabelRegionEXT(session);

```

In the above example, if an error occurred in the `// Do stuff 1` section, then your debug utils callback would contain the following data in its `sessionLabels` array:

- `[0]` = `individual_label` with `labelName` = "WaitFrame"
- `[1]` = `session_active_region_label` with `labelName` = "Session active"

However, if an error occurred in the `// Do stuff 2` section, then your debug utils callback would contain the following data in its `sessionLabels` array:

- `[0]` = `individual_label` with `labelName` = "BeginFrame"
- `[1]` = `session_frame_region_label` with `labelName` = "Session Frame 123"
- `[2]` = `session_active_region_label` with `labelName` = "Session active"

You'll notice that "WaitFrame" is no longer available as soon as the next call to another function like `xrSessionBeginDebugUtilsLabelRegionEXT`.

## Issues

None



## Version History

- Revision 1, 2018-02-19 (Mark Young / Karl Schultz)
  - Initial draft, based on `VK_EXT_debug_utils`.
- Revision 2, 2018-11-16 (Mark Young)
  - Clean up some language based on changes going into the Vulkan `VK_EXT_debug_utils` extension by Peter Kraus (aka @krOoze).
  - Added session labels
- Revision 3, 2019-07-19 (Rylie Pavlik)
  - Update examples.
  - Improve formatting.
- Revision 4, 2021-04-04 (Rylie Pavlik)
  - Fix missing error code.
  - Improve formatting.
- Revision 5, 2023-07-25 (John Kearney, Meta)
  - `XrDebugUtilsMessengerCallbackDataEXT` parameters `messageId` and `functionName` to be optional.

## 12.27. XR\_EXT\_dpad\_binding

### Name String

`XR_EXT_dpad_binding`

### Extension Type

Instance extension

### Registered Extension Number

79

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_KHR\\_binding\\_modification](#)

### Last Modified Date

2022-04-20

## IP Status

No known IP claims.

## Contributors

Joe Ludwig, Valve  
Keith Bradner, Valve  
Rune Berg, Valve  
Nathan Nuber, Valve  
Jakob Bornecrantz, Collabora  
Rylie Pavlik, Collabora  
Jules Blok, Epic Games

## Overview

This extension allows the application to bind one or more digital actions to a trackpad or thumbstick as though it were a dpad by defining additional component paths to suggest bindings for. The behavior of this dpad-like mapping may be customized using [XrInteractionProfileDpadBindingEXT](#).

Applications **must** also enable the [XR\\_KHR\\_binding\\_modification](#) extension that this builds on top of.

## New Component Paths

When this extension is enabled, a runtime **must** accept otherwise-valid suggested bindings that refer to the following component paths added to certain existing input source paths.

- For a given interaction profile,
  - For each [input source path](#) valid in that interaction profile that has identifier *trackpad* but without a component specified (i.e. `.../input/trackpad` or `.../input/trackpad_<location>`), a runtime **must** accept the following components appended to that path in a suggested binding:
    - `.../dpad_up`
    - `.../dpad_down`
    - `.../dpad_left`
    - `.../dpad_right`
    - `.../dpad_center`
  - For each [input source path](#) valid in that interaction profile that has identifier *thumbstick* but without a component specified (i.e. `.../input/thumbstick` or `.../input/thumbstick_<location>`), a runtime **must** accept the following components appended to that path in a suggested binding:
    - `.../dpad_up`
    - `.../dpad_down`
    - `.../dpad_left`
    - `.../dpad_right`

While a runtime **may** ignore accepted suggested bindings, and **may** use their contents as suggestions for automatic remapping when not obeying them, this extension defines interpretations the runtime **must** make in the case that a suggested binding using one of these paths is being obeyed.

An application **can** pass `XrInteractionProfileDpadBindingEXT` in the `XrBindingModificationsKHR::bindingModifications` array associated with a suggested binding to customize the behavior of this mapping in the case that suggested bindings are being obeyed, and to provide remapping hints in other cases. If no `XrInteractionProfileDpadBindingEXT` structure is present in `XrBindingModificationsKHR::bindingModifications` for a given action set and component-less input source path, the runtime **must** behave as if one were passed with the following values:

- `forceThreshold` = 0.5
- `forceThresholdReleased` = 0.4
- `centerRegion` = 0.5
- `wedgeAngle` =  $\frac{1}{2} \pi$
- `isSticky` = `XR_FALSE`
- `onHaptic` = `NULL`
- `offHaptic` = `NULL`

For the purposes of description, the (-1, 1) ranges of the  $x$  and  $y$  components of trackpad and thumbstick inputs are depicted in this extension as if their scale were equal between axes. However, this is **not** required by this extension: while their numeric scale is treated as equal, their physical scale **may** not be.

Each of the component paths defined by this extension behave as boolean inputs. The center component `.../dpad_center` (only present when the path identifier is `trackpad`) **must** not be active at the same time as any other dpad component. For the other components, zero, one, or (depending on the `wedgeAngle`) two of them **may** be active at any time, though only adjacent components on a single logical dpad may be active simultaneously. For example, `.../dpad_down` and `.../dpad_left` are adjacent, and thus **may** be active simultaneously, while `.../dpad_up` and `.../dpad_down` are not adjacent and **must** not be active simultaneously.

*Note*



If `wedgeAngle`  $> \frac{1}{2} \pi$ , it is possible for two components referring to adjacent directions (excluding `.../dpad_center`) to be active at the same time, as the directional regions overlap. If `wedgeAngle`  $< \frac{1}{2} \pi$ , there are wedges between directional regions that correspond to no dpad component.

The following components are defined by possibly-overlapping truncated wedges pointing away from 0, 0 in  $x, y$  input space, with their angular size of `XrInteractionProfileDpadBindingEXT::wedgeAngle` centered around the indicated direction.

- `.../dpad_up`: direction (0, 1)

- `.../dpad_down`: direction (0, -1)
- `.../dpad_left`: direction (-1, 0)
- `.../dpad_right`: direction (1, 0)

Typical values for `wedgeAngle` are  $\frac{1}{2} \pi$  (or 90°) for regions that do not overlap or  $\frac{3}{4} \pi$  (or 135°) for regions that are evenly divided between the exclusive region for one cardinal direction and the overlap with neighboring regions.

Each of these regions is truncated by an arc to exclude the area within a radius of `XrInteractionProfileDpadBindingEXT::centerRegion` away from 0, 0. When used with an input path with an identifier of `trackpad`, the area within this radius corresponds to the `.../dpad_center` component. When used with an input path with an identifier of `thumbstick`, the area within this radius is a region where all dpad components **must** be inactive.

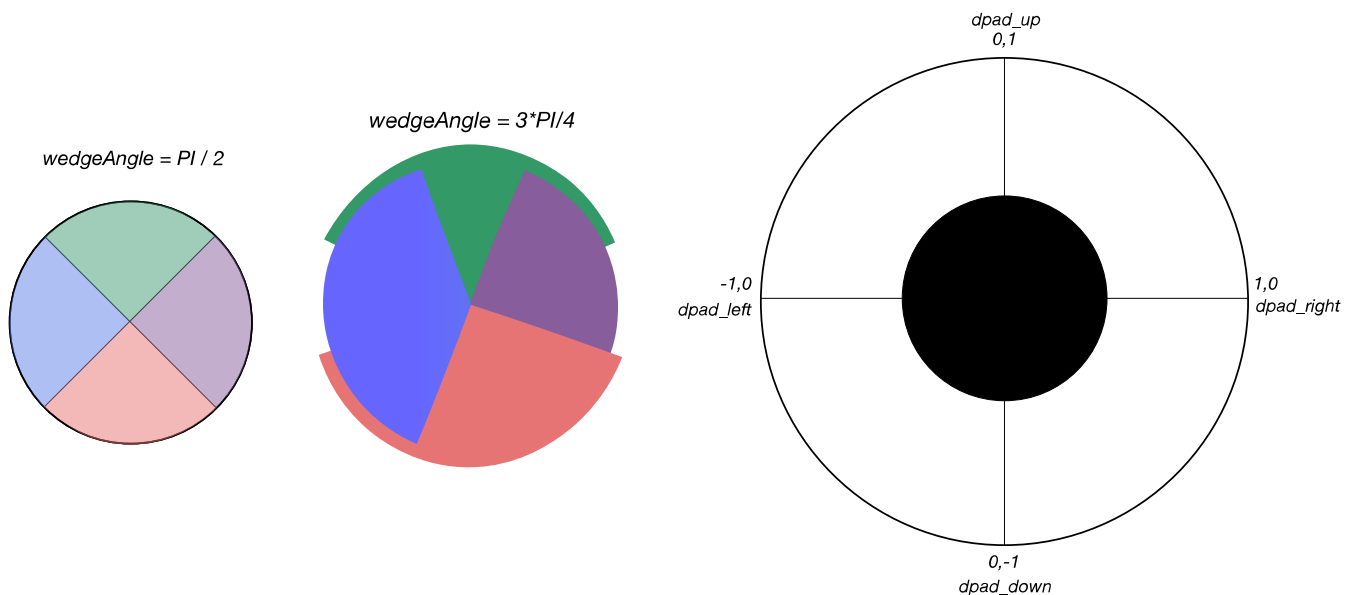


Figure 8. Wedge Angles

## Behavior

For both the `trackpad` and `thumbstick` input identifiers, there are conditions that **must** be true for any dpad component to report active. If these conditions are true, the selection of which component or components are active, if any, takes place.

- Activation of a dpad component when appended to an input path with identifier `trackpad` on the values of the `.../x` and `.../y` components, as well as on an overall activation state. If the overall state is inactive, the runtime **must** treat all corresponding dpad components as inactive.
  - If the component `.../click` is also valid for the trackpad, the overall activation state is equal to the value of the `.../click`.
  - If the component `.../click` is **not** valid for the trackpad, but the component `.../force` is valid, the overall activation state depends on the value of that `.../force` component, as well as the previous overall activation state for hysteresis. The `.../force` component value hysteresis thresholds for

overall activation are `XrInteractionProfileDpadBindingEXT::forceThreshold` and `forceThresholdReleased`. More explicitly:

- If the previous overall state was inactive, the current overall state **must** be active if and only if the value of the `.../force` component is greater than or equal to `forceThreshold`.
- If the previous overall state was active, the current state **must** be inactive if and only if the value of the `.../force` component is strictly less than `forceThresholdReleased`.
- Activation of a dpad component when appended to an input path with identifier `thumbstick` depends only on the value of the `.../x` and `.../y` components of that input.
  - If the thumbstick `x` and `y` values correspond to a deflection from center of less than `centerRegion`, all dpad components **must** be reported as inactive.

Hysteresis is desirable to avoid an unintentional, rapid toggling between the active and inactive state that can occur when the amount of force applied by the user is very close to the threshold at which the input is considered active. Hysteresis is optional, and is achieved through a difference between `forceThreshold` and `forceThresholdReleased`.

When `XrInteractionProfileDpadBindingEXT::isSticky` is `XR_FALSE`, and the above logic indicates that some dpad component is active, a runtime obeying suggested bindings **must** select which dpad components to report as active based solely on the current `x`, `y` values.

If `XrInteractionProfileDpadBindingEXT::isSticky` is `XR_TRUE`, the region(s) to be made active **must** be latched when the above logic begins to indicate that some dpad component is active, and the `x` and `y` values are within at least one region. The latched region(s) **must** continue to be reported as active until the activation logic indicates that all dpad components **must** be inactive. The latched region(s) remain active even if the input leaves that region or enters another region.

The runtime **must** latch the `x` and `y` values, and thus the region or regions (in the case of overlapping dpad component wedges), when the sticky activation toggle becomes true. The latched regions **must** continue to be true until the input returns to the center region (for a thumbstick) or is released (for a trackpad). In this way, sticky dpads maintain their selected region across touch/click transitions.

## Examples for `isSticky == XR_TRUE`

- Trackpad example: If the user clicks a trackpad in the `.../dpad_up` region, then (while clicked) slides their finger to the `.../dpad_down` region, `.../dpad_up` will remain true.
- Thumbstick example: If the user presses up on the thumbstick and activates the `.../dpad_up` region, then slides the thumbstick around to the `.../dpad_down` region without crossing the `centerRegion`, `.../dpad_up` is the virtual input that will be true.
- Thumbstick example: If the user presses up on the thumbstick and activates the `.../dpad_up` region, then slides the thumbstick directly down and through the region specified by `centerRegion` to `.../dpad_down`. Initially `.../dpad_up` will activate. Then when the thumbstick enters the `centerRegion` it will deactivate. Finally, when entering the `.../dpad_down` region `.../dpad_down` will activate.

## New Structures

The `XrInteractionProfileDpadBindingEXT` structure is defined as:

```
// Provided by XR_EXT_dpad_binding
typedef struct XrInteractionProfileDpadBindingEXT {
    XrStructureType      type;
    const void*          next;
    XrPath               binding;
    XrActionSet          actionSet;
    float                forceThreshold;
    float                forceThresholdReleased;
    float                centerRegion;
    float                wedgeAngle;
    XrBool32             isSticky;
    const XrHapticBaseHeader* onHaptic;
    const XrHapticBaseHeader* offHaptic;
} XrInteractionProfileDpadBindingEXT;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **binding** is the input path used for the specified actions in the suggested binding list to be used as a dpad. E.g. path:/user/hand/right/input/thumbstick
- **actionSet** is the action set for which this dpad will be active. The implementation **must** use the parameters from this structure for any actions from this action set that are bound to one of the dpad subpaths for this input path.
- **forceThreshold** a number in the half-open range (0, 1] representing the force value threshold at or above which ( $\geq$ ) a dpad input will transition from inactive to active.
- **forceThresholdReleased** a number in the half-open range (0, 1] representing the force value threshold strictly below which ( $<$ ) a dpad input will transition from active to inactive.
- **centerRegion** defines the center region of the thumbstick or trackpad. This is the radius, in the input value space, of a logically circular region in the center of the input, in the range (0, 1).
- **wedgeAngle** indicates the angle in radians of each direction region and is a value in the half-open range  $[0, \pi)$ .
- **isSticky** indicates that the implementation will latch the first region that is activated and continue to indicate that the binding for that region is true until the user releases the input underlying the virtual dpad.
- **onHaptic** is the haptic output that the runtime **must** trigger when the binding changes from false to true. If this field is `NULL`, the runtime **must** not trigger any haptic output on the threshold. This field **can** point to any supported sub-type of [XrHapticBaseHeader](#).
- **offHaptic** is the haptic output that the runtime **must** trigger when the binding changes from true to false. If this field is `NULL`, the runtime **must** not trigger any haptic output on the threshold. This field **can** point to any supported sub-type of [XrHapticBaseHeader](#).

The [XrInteractionProfileDpadBindingEXT](#) structure is an input struct that defines how to use any two-axis input to provide dpad-like functionality to the application. The struct **must** be added for each input that should be treated as a dpad to the [XrBindingModificationsKHR::bindingModifications](#) array in the [XrBindingModificationsKHR](#) structure (See [XR\\_KHR\\_binding\\_modification](#) extension).

Runtimes are free to ignore any of the fields when not obeying the bindings, but **may** use it for automatic rebindings of actions.

The implementation **must** return `XR_ERROR_VALIDATION_FAILURE` from [xrSuggestInteractionProfileBindings](#) if any of the following are true:

- **forceThreshold** or **forceThresholdReleased** are outside the half-open range (0, 1]

- `forceThreshold < forceThresholdReleased`
- `centerRegion` is outside the exclusive range (0, 1)
- `wedgeAngle` outside the half-open range [0,  $\pi$ )

If more than one `XrInteractionProfileDpadBindingEXT` is provided for the same input identifier, including top level path (e.g. `/user/hand/left/input/thumbstick`), and two or more of them specify the same actionset, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`. If the same input identifier, including top level path, is used for more than one action set, in addition to inputs being [suppressed by higher priority action sets](#), haptic events from dpads are also suppressed.

For example, a Valve Index controller binding with a "Walking" action set can have a dpad on each of:

- left thumbstick
- right thumbstick
- left trackpad
- right trackpad

Another action set can also have a dpad active on each of those inputs, and they can have different settings. If both action sets are active, the higher priority one trumps the lower priority one, and the lower priority one is suppressed.

### Valid Usage (Implicit)

- The `XR_EXT_dpad_binding` extension **must** be enabled prior to using `XrInteractionProfileDpadBindingEXT`
- `type` **must** be `XR_TYPE_INTERACTION_PROFILE_DPAD_BINDING_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `actionSet` **must** be a valid `XrActionSet` handle
- If `onHaptic` is not `NULL`, `onHaptic` **must** be a pointer to a valid `XrHapticBaseHeader`-based structure. See also: `XrHapticAmplitudeEnvelopeVibrationFB`, `XrHapticPcmVibrationFB`, `XrHapticVibration`
- If `offHaptic` is not `NULL`, `offHaptic` **must** be a pointer to a valid `XrHapticBaseHeader`-based structure. See also: `XrHapticAmplitudeEnvelopeVibrationFB`, `XrHapticPcmVibrationFB`, `XrHapticVibration`

## New Functions

### Issues

- What if an interaction profile is added that contains a `trackpad` identifier, for which there is neither a `.../click` or a `.../force` component?



- Equivalent logic would apply to whatever component is available to distinguish action from inaction.
- Is zero a valid wedge angle? Is  $\pi$ ?
  - Yes, though it is mostly useless, as it makes the directional regions empty in size and thus impossible to activate. The user could only activate `.../dpad_center` on a `trackpad` identifier.  $\pi$  is not a valid wedge angle because that would imply being able to activate three adjacent directions, of which two must be opposite. In practice, the sensors underlying these inputs make it effectively impossible to input an exact floating point value.

## Example

The following sample code shows how to create dpad bindings using this extension.

```

1 // Create dpad paths
2 XrPath pathThumbstick, pathDpadUp, pathDpadDown;
3 xrStringToPath( pInstance, "/user/hand/left/input/thumbstick", &pathThumbstick);
4 xrStringToPath( pInstance, "/user/hand/left/input/thumbstick/dpad_up", &pathDpadUp
  );
5 xrStringToPath( pInstance, "/user/hand/left/input/thumbstick/dpad_down",
  &pathDpadDown );
6
7 // Set dpad binding modifiers
8 XrInteractionProfileDpadBindingEXT xrDpadModification {
  XR_TYPE_INTERACTION_PROFILE_DPAD_BINDING_EXT };
9 xrDpadModification.actionSet = xrActionSet_Main;
10 xrDpadModification.binding = pathThumbstick;
11 xrDpadModification.centerRegion = 0.25f;
12 xrDpadModification.wedgeAngle = 2.0f;
13 // A gap between these next two members creates hysteresis, to avoid rapid
  toggling
14 xrDpadModification.forceThreshold = 0.8f;
15 xrDpadModification.forceThresholdReleased = 0.2f;
16
17 // Add dpad binding modifiers to binding modifications vector
18 std::vector< XrInteractionProfileDpadBindingEXT > vBindingModifs;
19 vBindingModifs.push_back( xrDpadModification );
20
21 std::vector< XrBindingModificationBaseHeaderKHR* > vBindingModifsBase;
22 for ( XrInteractionProfileDpadBindingEXT &modif : vBindingModifs )
23 {
24     vBindingModifsBase.push_back( reinterpret_cast<
  XrBindingModificationBaseHeaderKHR* >( &modif ) );
25 }
26
27 XrBindingModificationsKHR xrBindingModifications {
  XR_TYPE_BINDING_MODIFICATIONS_KHR };

```

```

28     xrBindingModifications.bindingModifications = vBindingModifsBase.data();
29     xrBindingModifications.bindingModificationCount = ( uint32_t )vBindingModifsBase
.size();
30
31     // Set dpad input path as suggested binding for an action
32     XrActionSuggestedBinding xrActionBindingTeleport, xrActionBindingMenu;
33
34     xrActionBindingTeleport.action = xrAction_Teleport;
35     xrActionBindingTeleport.binding = pathDpadUp;
36
37     xrActionBindingMenu.action = xrAction_Menu;
38     xrActionBindingMenu.binding = pathDpadDown;
39
40     std::vector< XrActionSuggestedBinding > vActionBindings;
41     vActionBindings.push_back( xrActionBindingTeleport );
42     vActionBindings.push_back( xrActionBindingMenu );
43
44     // Create interaction profile/controller path
45     XrPath xrInteractionProfilePath;
46     xrStringToPath( pInstance, "/interaction_profiles/valve/index_controller",
&xrInteractionProfilePath );
47
48     // Set suggested binding to interaction profile
49     XrInteractionProfileSuggestedBinding xrInteractionProfileSuggestedBinding {
XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING };
50     xrInteractionProfileSuggestedBinding.interactionProfile =
xrInteractionProfilePath;
51     xrInteractionProfileSuggestedBinding.suggestedBindings = vActionBindings.data();
52     xrInteractionProfileSuggestedBinding.countSuggestedBindings = ( uint32_t
)vActionBindings.size();
53
54     // Set binding modifications to interaction profile's suggested binding
55     xrInteractionProfileSuggestedBinding.next = &xrBindingModifications;
56
57     // Finally, suggest interaction profile bindings to runtime
58     xrSuggestInteractionProfileBindings( pInstance,
&xrInteractionProfileSuggestedBinding );

```

## Version History

- Revision 1, 2022-02-18 (Rune Berg)
  - Initial extension description

## 12.28. XR\_EXT\_eye\_gaze\_interaction

## Name String

`XR_EXT_eye_gaze_interaction`

## Extension Type

Instance extension

## Registered Extension Number

31

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2020-02-20

## IP Status

No known IP claims.

## Contributors

Denny Rönngren, Tobii  
Yin Li, Microsoft  
Alex Turner, Microsoft  
Paul Pedriana, Oculus  
Rémi Arnaud, Varjo  
Blake Taylor, Magic Leap  
Lachlan Ford, Microsoft  
Cass Everitt, Oculus

## Overview

This extension provides an [XrPath](#) for getting eye gaze input from an eye tracker to enable eye gaze interactions.

The intended use for this extension is to provide:

- system properties to inform if eye gaze interaction is supported by the current device.
- an [XrPath](#) for real time eye tracking that exposes an accurate and precise eye gaze pose to be used to enable eye gaze interactions.
- a structure [XrEyeGazeSampleTimeEXT](#) that allows for an application to retrieve more information regarding the eye tracking samples.

With these building blocks, an application can discover if the XR runtime has access to an eye tracker,

bind the eye gaze pose to the action system, determine if the eye tracker is actively tracking the users eye gaze, and use the eye gaze pose as an input signal to build eye gaze interactions.

### 12.28.1. Eye tracker

An eye tracker is a sensory device that tracks eyes and accurately maps what the user is looking at. The main purpose of this extension is to provide accurate and precise eye gaze for the application.

Eye tracking data can be sensitive personal information and is closely linked to personal privacy and integrity. It is strongly recommended that applications that store or transfer eye tracking data always ask the user for active and specific acceptance to do so.

If a runtime supports a permission system to control application access to the eye tracker, then the runtime **must** set the `isActive` field to `XR_FALSE` on the supplied `XrActionStatePose` structure, and **must** clear `XR_SPACE_LOCATION_POSITION_TRACKED_BIT`, `XR_SPACE_LOCATION_POSITION_VALID_BIT`, `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` when locating using the tracked space until the application has been allowed access to the eye tracker. When the application access has been allowed, the runtime **may** set `isActive` on the supplied `XrActionStatePose` structure to `XR_TRUE` and **may** set `XR_SPACE_LOCATION_POSITION_TRACKED_BIT`, `XR_SPACE_LOCATION_POSITION_VALID_BIT`, `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` when locating using the tracked space.

### 12.28.2. Device enumeration

When the eye gaze input extension is enabled an application **may** pass in a `XrSystemEyeGazeInteractionPropertiesEXT` structure in next chain structure when calling `xrGetSystemProperties` to acquire information about the connected eye tracker.

The runtime **must** populate the `XrSystemEyeGazeInteractionPropertiesEXT` structure with the relevant information to the `XrSystemProperties` returned by the `xrGetSystemProperties` call.

```
// Provided by XR_EXT_eye_gaze_interaction
typedef struct XrSystemEyeGazeInteractionPropertiesEXT {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsEyeGazeInteraction;
} XrSystemEyeGazeInteractionPropertiesEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsEyeGazeInteraction` the runtime **must** set this value to `XR_TRUE` when eye gaze sufficient for use cases such as aiming or targeting is supported by the current device, otherwise the runtime **must** set this to `XR_FALSE`.

## Valid Usage (Implicit)

- The `XR_EXT_eye_gaze_interaction` extension **must** be enabled prior to using `XrSystemEyeGazeInteractionPropertiesEXT`
- `type` **must** be `XR_TYPE_SYSTEM_EYE_GAZE_INTERACTION_PROPERTIES_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.28.3. Eye gaze input

This extension exposes a new interaction profile path `/interaction_profiles/ext/eye_gaze_interaction` that is valid for the user path

- `/user/eyes_ext`

for supported input source

- `.../input/gaze_ext/pose`

## Note

The interaction profile path `/interaction_profiles/ext/eye_gaze_interaction` defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called `/interaction_profiles/ext/eye_gaze_interaction_ext`, to allow for modifications when promoted to a KHR extension or the core specification.

The eye gaze pose is natively oriented with +Y up, +X to the right, and -Z forward and not gravity-aligned, similar to the `XR_REFERENCE_SPACE_TYPE_VIEW`. The eye gaze pose may originate from a point positioned between the user's eyes. At any point of time both the position and direction of the eye pose is tracked or untracked. This means that the runtime **must** set both `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` and `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` or clear both `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` and `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT`.

One particularity for eye trackers compared to most other spatial input is that the runtime may not have the capability to predict or interpolate eye gaze poses. Runtimes that cannot predict or interpolate eye gaze poses **must** clamp the gaze pose requested in the [xrLocateSpace](#) call to the value nearest to `time` requested in the call. To allow for an application to reason about high accuracy eye tracking, the application **can** chain in an [XrEyeGazeSampleTimeEXT](#) to the next pointer of the [XrSpaceLocation](#) structure passed into the [xrLocateSpace](#) call. The runtime **must** set `time` in the [XrEyeGazeSampleTimeEXT](#) structure to the clamped, predicted or interpolated time. The application **should** inspect the `time` field to understand when in time the pose is expressed. The `time` field **may** be in the future if a runtime can predict gaze poses. The runtime **must** set the `time` field to 0 if the sample time is not available.

When the runtime provides a nominal eye gaze pose, the `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` **must** be set if the eye otherwise has a fully-tracked pose relative to the other space. A runtime **can** provide a sub-nominal eye-gaze pose but **must** then clear the `XR_SPACE_LOCATION_POSITION_TRACKED_BIT`. An application can expect that a nominal eye gaze pose can be used for use cases such as aiming or targeting, while a sub-nominal eye gaze pose has degraded performance and should not be relied on for all input scenarios. Applications should be very careful when using sub-nominal eye gaze pose, since the behavior can vary considerably for different users and manufacturers, and some manufacturers **may** not provide sub-nominal eye gaze pose at all.

With current technology, some eye trackers **may** need to undergo an explicit calibration routine to provide a nominal accurate and precise eye gaze pose. If the eye tracker is in an uncalibrated state when the first call to [xrSyncActions](#) is made with an eye gaze action enabled, then the runtime **should** request eye tracker calibration from the user if it has not yet been requested.

```
// Provided by XR_EXT_eye_gaze_interaction
typedef struct XrEyeGazeSampleTimeEXT {
    XrStructureType    type;
    void*              next;
    XrTime              time;
} XrEyeGazeSampleTimeEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `time` is when in time the eye gaze pose is expressed.

## Valid Usage (Implicit)

- The `XR_EXT_eye_gaze_interaction` extension **must** be enabled prior to using `XrEyeGazeSampleTimeEXT`
- `type` **must** be `XR_TYPE_EYE_GAZE_SAMPLE_TIME_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.28.4. Sample code

The following example code shows how to bind the eye pose to the action system.

```
extern XrInstance instance;
extern XrSession session;
extern XrPosef pose_identity;

// Create action set
XrActionSetCreateInfo actionSetInfo{XR_TYPE_ACTION_SET_CREATE_INFO};
strcpy(actionSetInfo.actionSetName, "gameplay");
strcpy(actionSetInfo.localizedActionSetName, "Gameplay");
actionSetInfo.priority = 0;
XrActionSet gameplayActionSet;
CHK_XR(xrCreateActionSet(instance, &actionSetInfo, &gameplayActionSet));

// Create user intent action
XrActionCreateInfo actionInfo{XR_TYPE_ACTION_CREATE_INFO};
strcpy(actionInfo.actionName, "user_intent");
actionInfo.actionType = XR_ACTION_TYPE_POSE_INPUT;
strcpy(actionInfo.localizedActionName, "User Intent");
XrAction userIntentAction;
CHK_XR(xrCreateAction(gameplayActionSet, &actionInfo, &userIntentAction));

// Create suggested bindings
XrPath eyeGazeInteractionProfilePath;
CHK_XR(xrStringToPath(instance, "/interaction_profiles/ext/eye_gaze_interaction",
&eyeGazeInteractionProfilePath));

XrPath gazePosePath;
CHK_XR(xrStringToPath(instance, "/user/eyes_ext/input/gaze_ext/pose", &gazePosePath));

XrActionSuggestedBinding bindings;
bindings.action = userIntentAction;
bindings.binding = gazePosePath;

XrInteractionProfileSuggestedBinding suggestedBindings
```

```

{XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING};
suggestedBindings.interactionProfile = eyeGazeInteractionProfilePath;
suggestedBindings.suggestedBindings = &bindings;
suggestedBindings.countSuggestedBindings = 1;
CHK_XR(xrSuggestInteractionProfileBindings(instance, &suggestedBindings));

XrSessionActionSetsAttachInfo attachInfo{XR_TYPE_SESSION_ACTION_SETS_ATTACH_INFO};
attachInfo.countActionSets = 1;
attachInfo.actionSets = &gameplayActionSet;
CHK_XR(xrAttachSessionActionSets(session, &attachInfo));

XrActionSpaceCreateInfo createActionSpaceInfo{XR_TYPE_ACTION_SPACE_CREATE_INFO};
createActionSpaceInfo.action = userIntentAction;
createActionSpaceInfo.poseInActionSpace = pose_identity;
XrSpace gazeActionSpace;
CHK_XR(xrCreateActionSpace(session, &createActionSpaceInfo, &gazeActionSpace));

XrReferenceSpaceCreateInfo createReferenceSpaceInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};
createReferenceSpaceInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_LOCAL;
createReferenceSpaceInfo.poseInReferenceSpace = pose_identity;
XrSpace localReferenceSpace;
CHK_XR(xrCreateReferenceSpace(session, &createReferenceSpaceInfo, &localReferenceSpace));

while(true)
{
    XrActiveActionSet activeActionSet{gameplayActionSet, XR_NULL_PATH};
    XrTime time;

    XrActionsSyncInfo syncInfo{XR_TYPE_ACTIONS_SYNC_INFO};
    syncInfo.countActiveActionSets = 1;
    syncInfo.activeActionSets = &activeActionSet;
    CHK_XR(xrSyncActions(session, &syncInfo));

    XrActionStatePose actionStatePose{XR_TYPE_ACTION_STATE_POSE};
    XrActionStateGetInfo getActionStateInfo{XR_TYPE_ACTION_STATE_GET_INFO};
    getActionStateInfo.action = userIntentAction;
    CHK_XR(xrGetActionStatePose(session, &getActionStateInfo, &actionStatePose));

    if(actionStatePose.isActive){
        XrEyeGazeSampleTimeEXT eyeGazeSampleTime{XR_TYPE_EYE_GAZE_SAMPLE_TIME_EXT};
        XrSpaceLocation gazeLocation{XR_TYPE_SPACE_LOCATION, &eyeGazeSampleTime};
        CHK_XR(xrLocateSpace(gazeActionSpace, localReferenceSpace, time, &gazeLocation));

        // Do things
    }
}

```



## Version History

- Revision 1, 2020-02-20 (Denny Rönngren)
  - Initial version
- Revision 2, 2022-05-27 (Bryce Hutchings)
  - Remove error-prone `XrEyeGazeSampleTimeEXT` validation requirement

## 12.29. XR\_EXT\_future

### Name String

`XR_EXT_future`

### Extension Type

Instance extension

### Registered Extension Number

470

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Bryce Hutchings, Microsoft  
Andreas Selvik, Meta  
Ron Bessems, Magic Leap  
Yin Li, Microsoft Corporation  
Baolin Fu, Bytedance  
Cass Everitt, Meta Platforms  
Charlton Rodda, Collabora  
Jakob Bornecrantz, NVIDIA  
John Kearney, Meta Platforms  
Jonathan Wright, Meta Platforms  
Jun Yan, ByteDance  
Junyi Wang, Bytedance Ltd.  
Karthik Kadappan, Magic Leap  
Natalie Fleury, Meta Platforms  
Nathan Nuber, Valve  
Nikita Lutsenko, Meta Platforms  
Robert Blenkinsopp, Ultraleap  
Rylie Pavlik, Collabora

Tim Mowrer, Meta Platforms  
Wenlin Mao, Meta Platforms  
Will Fu, Bytedance  
Zhipeng Liu, Bytedance

### 12.29.1. Overview

In XR systems there are certain operations that are long running and do not reasonably complete within a normal frame loop. This extension introduces the concept of a *future* which supports creation of asynchronous (async) functions for such long running operations. This extension does not include any asynchronous operations: it is expected that other extensions will use these *futures* and their associated conventions in this extension to define their asynchronous operations.

An `XrFutureEXT` represents the future result of an asynchronous operation, comprising an `XrResult` and possibly additional outputs. Long running operations immediately return an `XrFutureEXT` when started, letting the application poll the state of the future, and get the result once ready by calling a "complete"-function.

### 12.29.2. Getting a future

The `XrFutureEXT` basetype is defined as:

```
// Provided by XR_EXT_future
XR_DEFINE_OPAQUE_64(XrFutureEXT)
```

Asynchronous functions return an `XrFutureEXT` token as a placeholder for a value that will be returned later. An `XrFutureEXT` returned by a successful call to a function starting an asynchronous operation **should** normally start in the `XR_FUTURE_STATE_PENDING_EXT` state, but **may** skip directly to `XR_FUTURE_STATE_READY_EXT` if the result is immediately available.

The value `XR_NULL_FUTURE_EXT`, numerically equal to `0`, is never a valid `XrFutureEXT` value.

**Note** that an `XrFutureEXT` token is neither a `handle` nor an `atom` type (such as `XrPath`). It belongs to a new category and is defined as an opaque 64-bit value. See [Future Scope](#) for details on the scope and lifecycle of a future.

**Style note:** Functions that return an `XrFutureEXT` **should** be named with the suffix "Async", e.g. `xrPerformLongTaskAsync`. This function **must** not set the `XrFutureEXT` to `XR_NULL_FUTURE_EXT` when the function returns `XR_SUCCESS`.

### 12.29.3. Waiting for a future to become ready

The `xrPollFutureEXT` function is defined as:

```
// Provided by XR_EXT_future
XrResult xrPollFutureEXT(
    XrInstance                instance,
    const XrFuturePollInfoEXT* pollInfo,
    XrFuturePollResultEXT*    pollResult);
```

## Parameter Descriptions

- `instance` is an `XrInstance` handle
- `pollInfo` is a pointer to an `XrFuturePollInfoEXT` structure.
- `pollResult` is a pointer to an `XrFuturePollResultEXT` structure to be populated on a successful call.

Applications **can** use this function to check the current state of a future, typically while waiting for the async operation to complete and the future to become "ready" to complete.



### Note

Each `XrFutureEXT` value **must** be externally synchronized by the application when calling completion, polling, and cancellation functions, and when destroying the associated handle.

## Valid Usage (Implicit)

- The `XR_EXT_future` extension **must** be enabled prior to calling `xrPollFutureEXT`
- `instance` **must** be a valid `XrInstance` handle
- `pollInfo` **must** be a pointer to a valid `XrFuturePollInfoEXT` structure
- `pollResult` **must** be a pointer to an `XrFuturePollResultEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_FUTURE_INVALID_EXT`

The `XrFuturePollInfoEXT` structure is defined as:

```
// Provided by XR_EXT_future
typedef struct XrFuturePollInfoEXT {
    XrStructureType    type;
    const void*       next;
    XrFutureEXT        future;
} XrFuturePollInfoEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `future` is the `XrFutureEXT` future being polled.

An `XrFuturePollInfoEXT` structure is used to pass `future` to `xrPollFutureEXT`.

## Valid Usage (Implicit)

- The `XR_EXT_future` extension **must** be enabled prior to using `XrFuturePollInfoEXT`
- `type` **must** be `XR_TYPE_FUTURE_POLL_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`

The `XrFuturePollResultEXT` structure is defined as:

```
// Provided by XR_EXT_future
typedef struct XrFuturePollResultEXT {
    XrStructureType    type;
    void*              next;
    XrFutureStateEXT   state;
} XrFuturePollResultEXT;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `state` is the `XrFutureStateEXT` of the `XrFutureEXT` passed to `xrPollFutureEXT`.

An `XrFuturePollResultEXT` structure is used to return the result of `xrPollFutureEXT`.

### Valid Usage (Implicit)

- The `XR_EXT_future` extension **must** be enabled prior to using `XrFuturePollResultEXT`
- `type` **must** be `XR_TYPE_FUTURE_POLL_RESULT_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 12.29.4. Completing a Future

Extensions that provide async functions returning a future **should** also provide a matching completion function to "complete" the future in order to return the result of the asynchronous operation. This function **should** be named with the suffix "Complete" replacing the "Async" suffix, e.g. `xrPerformLongTaskComplete` is a suitable completion function name corresponding to `xrPerformLongTaskAsync`.

A completion function **must** populate a structure that **must** be based on `XrFutureCompletionBaseHeaderEXT` to return the result of the asynchronous operation. Such a structure **may** be `static_cast` to and from `XrFutureCompletionBaseHeaderEXT`, allowing generic handling of the asynchronous operation results as well as polymorphic output from such an operation. The `XrResult` returned from a completion function **must** not be used to return the result of the asynchronous operation. Instead, the `XrResult` returned from a completion function **must** indicate both whether the completion function was called correctly, and if the completion of the future succeeded.

For instance, a completion function returning `XR_ERROR_HANDLE_INVALID` means that a handle passed to the completion function was invalid, not that a handle associated with the asynchronous operation is invalid. Note that `XR_SUCCESS` **should** be returned from the completion function even if the asynchronous operation itself was a failure; that failure is indicated in `XrFutureCompletionBaseHeaderEXT::futureResult` rather than the return value of the completion function.

When a completion function is called with a future that is in the `XR_FUTURE_STATE_PENDING_EXT` state, the runtime **must** return `XR_ERROR_FUTURE_PENDING_EXT`.

The `XrResult` of the asynchronous operation **must** be returned in the `futureResult` of the return structure extending `XrFutureCompletionBaseHeaderEXT`. Completion functions which only need to return an `XrResult` **may** populate the `XrFutureCompletionEXT` structure provided by this extension as their output structure.

Once a completion function is called on a future with a valid output structure and returns `XR_SUCCESS`, the future is considered **completed**, and therefore **invalidated**. Any usage of this future thereafter **must** return `XR_ERROR_FUTURE_INVALID_EXT`.

Passing a completed future to any function accepting futures **must** return `XR_ERROR_FUTURE_INVALID_EXT`.

The runtime **may** release any resources associated with an `XrFutureEXT` once the future has been completed or invalidated.

*Note*



Each `XrFutureEXT` value **must** be externally synchronized by the application when calling completion, polling, and cancellation functions, and when destroying the associated handle.

The `XrFutureCompletionBaseHeaderEXT` structure is defined as:

```
// Provided by XR_EXT_future
typedef struct XrFutureCompletionBaseHeaderEXT {
    XrStructureType    type;
    void*              next;
    XrResult            futureResult;
} XrFutureCompletionBaseHeaderEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `futureResult` is [XrResult](#) of the async operation associated with future passed to the completion function.

[XrFutureCompletionBaseHeaderEXT](#) is a base header for the result of a future completion function.

## Valid Usage (Implicit)

- The `XR_EXT_future` extension **must** be enabled prior to using [XrFutureCompletionBaseHeaderEXT](#)
- `type` **must** be `XR_TYPE_FUTURE_COMPLETION_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `futureResult` **must** be a valid [XrResult](#) value

The [XrFutureCompletionEXT](#) structure is defined as:

```
// Provided by XR_EXT_future
typedef struct XrFutureCompletionEXT {
    XrStructureType    type;
    void*              next;
    XrResult           futureResult;
} XrFutureCompletionEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `futureResult` is [XrResult](#) of the async operation associated with future passed to the completion function.

This is a minimal implementation of [XrFutureCompletionBaseHeaderEXT](#), containing only the fields present in the base header structure. It is intended for use by asynchronous operations that do not have other outputs or return values beyond an [XrResult](#) value, as the output parameter of their

completion function.

### Valid Usage (Implicit)

- The `XR_EXT_future` extension **must** be enabled prior to using `XrFutureCompletionEXT`
- `type` **must** be `XR_TYPE_FUTURE_COMPLETION_EXT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `futureResult` **must** be a valid `XrResult` value

## 12.29.5. Two-Call Idiom in Asynchronous Operations

OpenXR uses a `two-call idiom` for interfaces that return arrays or buffers of variable size. Asynchronous operations returning such an array or buffer similarly use the structure style of that two-call idiom, with small modifications to the typical completion function conventions to account for this pattern.

For completion functions returning an array or buffer using the two-call idiom, the future **must** be marked as **completed** if the output array size is sufficient for all elements of the data and was thus populated by the completion function. If the output array size is not sufficient, the runtime **must** not mark the future as completed nor invalidated.

For an array of zero data elements, this means the first call to the two-call idiom completion function **must** mark the future as **completed** and invalidated, even if the array is a `NULL` pointer. If `XrFutureCompletionBaseHeaderEXT::futureResult` is a `failure` the runtime **must invalidate** the future after the first call, and any further usage of this future **must** return `XR_ERROR_FUTURE_INVALID_EXT`.

For non-zero output arrays where `XrFutureCompletionBaseHeaderEXT::futureResult` is not a failure, `XrFutureCompletionBaseHeaderEXT::futureResult` **must** be identical for both calls to the completion function.

This definition allows asynchronous operations to return dynamically sized outputs by using the `two-call idiom` in a familiar way.

## 12.29.6. Cancelling a future

The `xrCancelFutureEXT` function is defined as:

```
// Provided by XR_EXT_future
XrResult xrCancelFutureEXT(
    XrInstance          instance,
    const XrFutureCancelInfoEXT* cancelInfo);
```



## Parameter Descriptions

- `instance` is an [XrInstance](#) handle
- `cancelInfo` is a pointer to an [XrFutureCancelInfoEXT](#) structure.

This function cancels the future and signals that the async operation is not required. After a future has been cancelled any functions using this future **must** return `XR_ERROR_FUTURE_INVALID_EXT`.

A runtime **may** stop the asynchronous operation associated with a future after an app has cancelled it.



### Note

Each [XrFutureEXT](#) value **must** be externally synchronized by the application when calling completion, polling, and cancellation functions, or destroying the associated handle.

## Valid Usage (Implicit)

- The `XR_EXT_future` extension **must** be enabled prior to calling [xrCancelFutureEXT](#)
- `instance` **must** be a valid [XrInstance](#) handle
- `cancelInfo` **must** be a pointer to a valid [XrFutureCancelInfoEXT](#) structure

## Thread Safety

- Access to the `future` member of the `cancelInfo` parameter **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_FUTURE_INVALID_EXT`

The [XrFutureCancelInfoEXT](#) structure is defined as:

```
// Provided by XR_EXT_future
typedef struct XrFutureCancelInfoEXT {
    XrStructureType    type;
    const void*        next;
    XrFutureEXT        future;
} XrFutureCancelInfoEXT;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `future` is [XrFutureEXT](#) to cancel.

An [XrFutureCancelInfoEXT](#) describes which future to cancel.

### Valid Usage (Implicit)

- The [XR\\_EXT\\_future](#) extension **must** be enabled prior to using [XrFutureCancelInfoEXT](#)
- `type` **must** be `XR_TYPE_FUTURE_CANCEL_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 12.29.7. XrFutureEXT Lifecycle

The [XrFutureStateEXT](#) enumerates the possible future lifecycle states:

```
// Provided by XR_EXT_future
typedef enum XrFutureStateEXT {
    XR_FUTURE_STATE_PENDING_EXT = 1,
    XR_FUTURE_STATE_READY_EXT = 2,
    XR_FUTURE_STATE_MAX_ENUM_EXT = 0x7FFFFFFF
} XrFutureStateEXT;
```

## Enumerant Descriptions

- `XR_FUTURE_STATE_PENDING_EXT`. The state of a future that is waiting for the async operation to conclude. This is typically the initial state of a future returned from an async function.
- `XR_FUTURE_STATE_READY_EXT`. The state of a future when the result of the async operation is ready. The application **can** retrieve the result by calling the associated completion function.

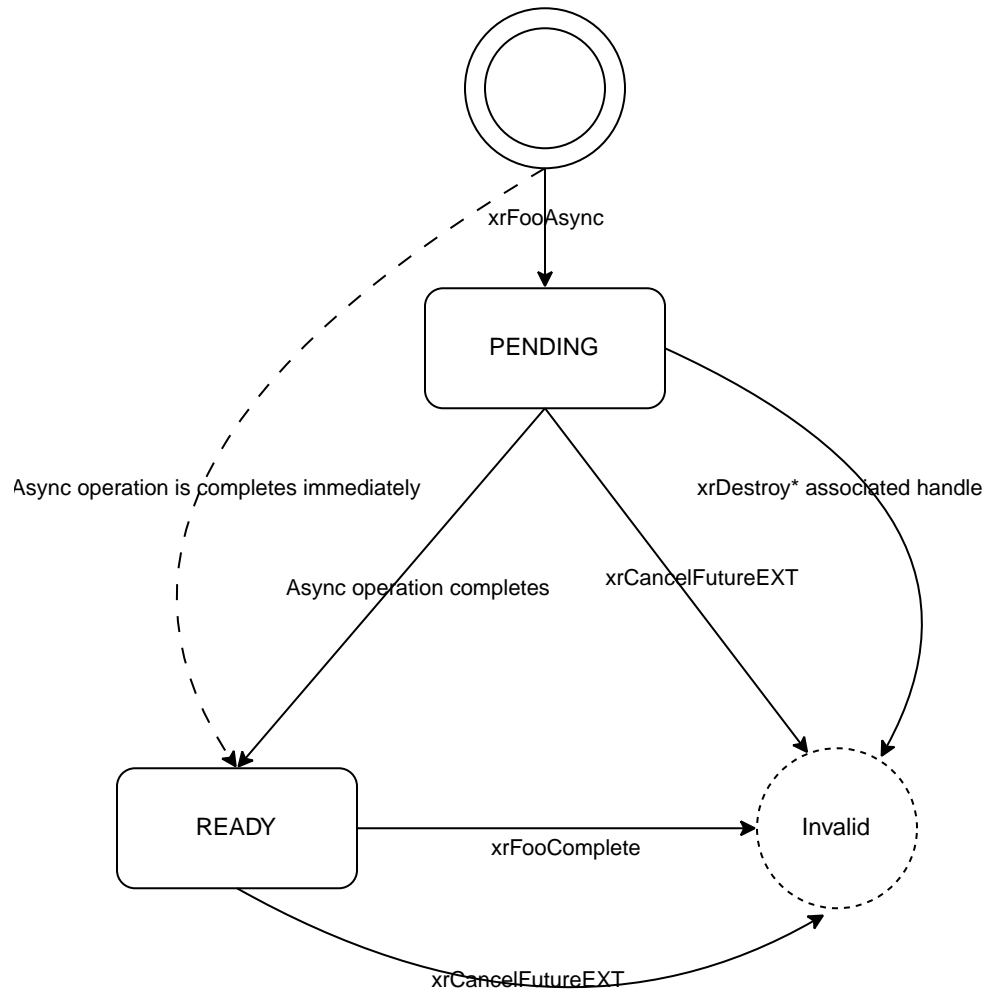


Figure 9. `XrFutureEXT` Life-cycle

A future that is not invalidated (or completed) **may** be in one of two states, **Pending** and **Ready**, represented by `XR_FUTURE_STATE_PENDING_EXT` and `XR_FUTURE_STATE_READY_EXT` respectively.

- When successfully returned from an async function the future starts out as **Pending**. In this state the future **may** be polled, but **must** not be passed to a completion function. Applications **should** wait for the future to become ready and keep polling the state of the future. If a pending future is passed to the associated completion function, it **must** return `XR_ERROR_FUTURE_PENDING_EXT`.
- Once the asynchronous operation succeeds or fails, the state of the future moves to **Ready**. In the ready state the future **may** be "Completed" with the `Complete` function. See [Completing a Future](#).
- After being successfully completed, the future becomes invalidated if the completion function

returns a success code, and in the case of two-call idioms, the array was not `NULL`.

- After a call to `xrCancelFutureEXT`, the future becomes invalidated immediately and any resources associated with it **may** be freed (including handles)
- When the associated handle is destroyed, the futures become invalidated. See [Future Scope](#).

A future returned from an async function **must** be in either the state `XR_FUTURE_STATE_PENDING_EXT` or `XR_FUTURE_STATE_READY_EXT`. A runtime **may** skip the `Pending` state and go directly to `Ready` if the result is immediately available.

## 12.29.8. Future Scope

An `XrFutureEXT` is scoped to the "associated handle" of the future. The associated handle is the handle passed to the asynchronous operation that returns the `XrFutureEXT`. When the associated handle is destroyed, the runtime **must** invalidate the future and **may** free any associated resources.

### Note



For example, for a hypothetical async function `xrGetFooAsync(Session session, XrFooGetInfo info, XrFutureEXT* future)` then `XrSession` is the associated handle, and if the app calls `xrDestroySession(...)` the returned future becomes invalid.

Likewise, for `xrRequestBar(BarGenerator barGenerator, XrBarGenerateInfo info, XrFutureEXT* future)`, the hypothetical `BarGenerator` is the associated handle that scopes the future.

## 12.29.9. Extension Guidelines for Asynchronous Functions

Extensions exposing asynchronous functions using `XR_EXT_future` **should** follow the following patterns:

1. Functions returning a future **should** use the suffix "Async", prior to an author/vendor tag if applicable. For example:
  - `xrGetFooAsync(...)`
  - `xrRequestBarAsyncKHR(...)`
  - `xrCreateObjectAsyncVENDOR(...)`
2. The name of the future out parameter **should** be `future`. For example:
  - `xrGetFooAsync(..., XrFutureEXT* future)`
  - `xrRequestBarAsyncKHR(..., XrFutureEXT* future)`
  - `xrCreateObjectAsyncVENDOR(..., XrFutureEXT* future)`
3. Functions completing a future **should** match the name of the function returning the future, but with "Complete" rather than "Async" as the suffix. This is a deviation from the normal pattern in OpenXR, if "complete" is considered to be the verb; however this provides for a useful sorting order keeping the "Async" and "Complete" functions adjacent, and fits the pattern of using suffixes for

asynchronous functions. The completion function **must** use the same handle type as the corresponding async function and the runtime **must** return `XR_ERROR_HANDLE_INVALID` if the handle value passed to the completion function is different from the value passed to the async function that returned the future. For example:

- `xrGetFooComplete(...)`
- `xrRequestBarCompleteKHR(...)`,
- `xrCreateObjectCompleteVENDOR(...)`

4. The output structure used in the "Complete" function **should** extend `XrFutureCompletionBaseHeaderEXT` (starting with `type`, `next`, and `futureResult` fields).

5. If an operation requires more than the basic `XrFutureCompletionEXT` output, the output structure populated by the "Complete" function **should** be named based on the function that returned the future, with the suffix "Completion". For example:

- `xrGetFooComplete` populates `XrGetFooCompletion`
- `xrRequestBarComplete` populates `XrRequestBarCompletionKHR`
- `xrCreateObjectCompleteVENDOR` populates `XrCreateObjectCompletionVENDOR`

6. The `XrFutureEXT` parameter in the "Complete" function **should** be named `future`. For example:

- `xrGetFooComplete(..., XrFutureEXT future)`
- `xrRequestBarCompleteKHR(..., XrFutureEXT future)`
- `xrCreateObjectCompleteVENDOR(..., XrFutureEXT future)`

7. The parameter with the completion structure **should** be named `completion`. e.g.

- `xrGetFooComplete(..., XrFutureEXT future, XrGetFooCompletion* completion)`
- `xrRequestBarCompleteKHR(..., XrFutureEXT future, XrRequestBarCompletionKHR* completion)`
- `xrCreateObjectCompleteVENDOR(..., XrFutureEXT future, XrCreateObjectCompletionVENDOR* completion)`

## 12.29.10. Asynchronous function patterns

### `xrCreate` functions

```
/* ***** */
/* Foo extension definition */
/* ***** */
typedef void *XrFoo; // Handle definition
typedef struct XrFooObjectCreateInfo {
    XrStructureType type;
    const void *next;
} XrFooObjectCreateInfo;
#define XR_TYPE_FOO_OBJECT_CREATE_INFO ((XrStructureType)1100092000U)
```

```

// extends struct XrFutureCompletionBaseHeader using "parentstruct"
typedef struct XrFooObjectCreateCompletionEXT {
    XrStructureType type;
    void *XR_MAY_ALIAS next;
    XrResult futureResult;
    XrFoo foo;
} XrFooObjectCreateCompletionEXT;
#define XR_TYPE_FOO_OBJECT_CREATE_COMPLETION ((XrStructureType)1100092001U)

typedef XrResult(XRAPI_PTR *PFN_xrCreateFooObjectAsync)(
    XrSession session, const XrFooObjectCreateInfo *createInfo,
    XrFutureEXT *future);
typedef XrResult(XRAPI_PTR *PFN_xrCreateFooObjectComplete)(
    XrSession session, XrFutureEXT future,
    XrFooObjectCreateCompletionEXT *completion);

/*****/
/* End Foo definition */
/*****/

PFN_xrCreateFooObjectAsync xrCreateFooObjectAsync; // previously initialized
PFN_xrCreateFooObjectComplete
    xrCreateFooObjectComplete; // previously initialized
PFN_xrPollFutureEXT xrPollFutureEXT; // previously initialized
XrInstance instance; // previously initialized
XrSession session; // previously initialized

XrFutureEXT futureFooObject;
XrResult result;

XrFooObjectCreateInfo createInfo{XR_TYPE_FOO_OBJECT_CREATE_INFO};
result = xrCreateFooObjectAsync(session, &createInfo, &futureFooObject);
CHK_XR(result);

bool keepLooping = true;
bool futureReady = false;
while (keepLooping) {
    XrFuturePollInfoEXT pollInfo{XR_TYPE_FUTURE_POLL_INFO_EXT};
    XrFuturePollResultEXT pollResult{XR_TYPE_FUTURE_POLL_RESULT_EXT};
    pollInfo.future = futureFooObject;
    CHK_XR(xrPollFutureEXT(instance, &pollInfo, &pollResult));

    if (pollResult.state == XR_FUTURE_STATE_READY_EXT) {
        futureReady = true;
        keepLooping = false;
    } else {
        // sleep(10);
    }
}

```

```

}

if (futureReady) {
    XrFooObjectCreateCompletionEXT completion{
        XR_TYPE_FOO_OBJECT_CREATE_COMPLETION};
    result = xrCreateFooObjectComplete(session, futureFooObject, &completion);
    CHK_XR(result); // Result of the complete function
    CHK_XR(completion.futureResult); // Return code of the create function
    // completion.fooObject is now valid and may be used!
}

```

## Two-call idiom

```

/*****/
/* Foo extension definition */
/*****/
typedef struct XrFooObjectCreateInfo {
    XrStructureType type;
    const void *next;
} XrFooObjectCreateInfo;
#define XR_TYPE_FOO_OBJECTS_CREATE_INFO ((XrStructureType)1100092002U)

// extends struct XrFutureCompletionBaseHeader using "parentstruct"
typedef struct XrFooObjectsCreateCompletionEXT {
    XrStructureType type;
    void *next;
    XrResult futureResult;
    uint32_t elementCapacityInput;
    uint32_t elementCapacityOutput;
    float *elements;
} XrFooObjectsCreateCompletionEXT;
#define XR_TYPE_FOO_OBJECTS_CREATE_COMPLETION ((XrStructureType)1100092003U)

typedef XrResult(XRAPI_PTR *PFN_xrCreateFooObjectsAsync)(
    XrSession session, const XrFooObjectCreateInfo *createInfo,
    XrFutureEXT *future);
typedef XrResult(XRAPI_PTR *PFN_xrCreateFooObjectsComplete)(
    XrSession session, XrFutureEXT future,
    XrFooObjectsCreateCompletionEXT *completion);

/*****/
/* End Foo definition */
/*****/

PFN_xrCreateFooObjectsAsync xrCreateFooObjectsAsync; // previously initialized
PFN_xrCreateFooObjectsComplete

```

```

    xrCreateFooObjectsComplete; // previously initialized
PFN_xrPollFutureEXT xrPollFutureEXT; // previously initialized
XrInstance instance; // previously initialized
XrSession session; // previously initialized

XrFutureEXT futureFooObjects;
XrResult result;

XrFooObjectCreateInfo createInfo{XR_TYPE_FOO_OBJECTS_CREATE_INFO};
result = xrCreateFooObjectsAsync(session, &createInfo, &futureFooObjects);
CHK_XR(result);

bool keepLooping = true;
bool futureReady = false;
while (keepLooping) {
    XrFuturePollInfoEXT pollInfo{XR_TYPE_FUTURE_POLL_INFO_EXT};
    XrFuturePollResultEXT pollResult{XR_TYPE_FUTURE_POLL_RESULT_EXT};
    pollInfo.future = futureFooObjects;
    CHK_XR(xrPollFutureEXT(instance, &pollInfo, &pollResult));

    if (pollResult.state == XR_FUTURE_STATE_READY_EXT) {
        futureReady = true;
        keepLooping = false;
    } else {
        // sleep(10);
    }
}

if (futureReady) {
    XrFooObjectsCreateCompletionEXT completion{
        XR_TYPE_FOO_OBJECTS_CREATE_COMPLETION};
    result = xrCreateFooObjectsComplete(session, futureFooObjects, &completion);
    CHK_XR(result); // Result of the complete function
    CHK_XR(completion.futureResult);

    std::vector<float> floatValues(completion.elementCapacityOutput);
    completion.elementCapacityInput = (uint32_t)floatValues.size();
    completion.elements = floatValues.data();

    result = xrCreateFooObjectsComplete(session, futureFooObjects, &completion);
    CHK_XR(result); // Result of the complete function
}

// completion.elements has now been filled with values by the runtime.

```



## Sample code

```
/* *****  
/* Slow Foo extension definition */  
/* *****  
// extends struct XrFutureCompletionBaseHeader using "parentstruct"  
typedef struct XrSlowFooCompletionEXT {  
    XrStructureType type;  
    void *XR_MAY_ALIAS next;  
    XrResult futureResult;  
    float foo;  
} XrSlowFooCompletionEXT;  
#define XR_TYPE_SLOW_FOO_COMPLETION_EXT ((XrStructureType)1100092005U)  
  
typedef struct XrSlowFooInfoEXT {  
    XrStructureType type;  
    void *XR_MAY_ALIAS next;  
} XrSlowFooInfoEXT;  
#define XR_TYPE_SLOW_FOO_INFO_EXT ((XrStructureType)1100092006U)  
  
typedef XrResult(XR_API_PTR *PFN_xrSlowFooAsyncEXT)(XrSession session,  
                                                    XrSlowFooInfoEXT slowFooInfo,  
                                                    XrFutureEXT *future);  
  
typedef XrResult(XR_API_PTR *PFN_xrSlowFooCompleteEXT)(  
    XrSession session, XrFutureEXT future, XrSlowFooCompletionEXT *completion);  
  
/* *****  
/* End Slow Foo extension definition */  
/* *****  
  
class MyGame {  
    void OnSlowFooRequest() {  
        if (m_slowFooFuture == XR_NULL_FUTURE_EXT) {  
            // Make initial request.  
            XrSlowFooInfoEXT fooInfo{XR_TYPE_SLOW_FOO_INFO_EXT};  
            XrResult result = xrSlowFooAsyncEXT(session, fooInfo, &m_slowFooFuture);  
            CHK_XR(result);  
        }  
    }  
  
    void OnGameTickOrSomeOtherReoccurringFunction() {  
  
        // Check if a future is outstanding  
        if (m_slowFooFuture == XR_NULL_FUTURE_EXT) {  
            return;  
        }  
    }  
}
```

```

// Poll for state of future
XrFuturePollInfoEXT pollInfo{XR_TYPE_FUTURE_POLL_INFO_EXT};
XrFuturePollResultEXT pollResult{XR_TYPE_FUTURE_POLL_RESULT_EXT};
pollInfo.future = m_slowFooFuture;
CHK_XR(xrPollFutureEXT(instance, &pollInfo, &pollResult));

if (pollResult.state == XR_FUTURE_STATE_READY_EXT) {
    // Complete the future, consuming the result
    XrSlowFooCompletionEXT completion{XR_TYPE_SLOW_FOO_COMPLETION_EXT};
    XrResult result =
        xrSlowFooCompleteEXT(session, m_slowFooFuture, &completion);
    // Check XrResult from the completion function
    CHK_XR(result);
    // Check XrResult from the async operation
    CHK_XR(completion.futureResult);
    m_fooValue = completion.foo;
    m_slowFooFuture = XR_NULL_FUTURE_EXT;
}
}

XrFutureEXT m_slowFooFuture{XR_NULL_FUTURE_EXT};
float m_fooValue{0.0f};

PFN_xrSlowFooAsyncEXT xrSlowFooAsyncEXT; // previously initialized
PFN_xrSlowFooCompleteEXT xrSlowFooCompleteEXT; // previously initialized
PFN_xrPollFutureEXT xrPollFutureEXT; // previously initialized
XrInstance instance; // previously initialized
XrSession session; // previously initialized
};

```

## Multi-threaded code

```

class MyThreadedGame {

MyThreadedGame() {
    // Start the thread
    m_processThread = std::thread(&MyThreadedGame::ThreadFunction, this);
    StartSlowFooRequest();
}

~MyThreadedGame() {
    // all functions using futures must be synchronized.
    CancelSlowFooRequestFuture();
    m_abort = true;
    m_processThread.join();
}
}

```

```

void StartSlowFooRequest() {
    std::unique_lock<std::mutex> lock(m_mutex);
    if (m_slowFooFuture == XR_NULL_FUTURE_EXT) {
        // Make initial request.
        XrSlowFooInfoEXT fooInfo{XR_TYPE_SLOW_FOO_INFO_EXT};
        XrResult result = xrSlowFooAsyncEXT(session, fooInfo, &m_slowFooFuture);
        CHK_XR(result);
    }
}

void CancelSlowFooRequestFuture() {
    std::unique_lock<std::mutex> lock(m_mutex);
    if (m_slowFooFuture != XR_NULL_FUTURE_EXT) {
        XrFutureCancelInfoEXT cancel_info{XR_TYPE_FUTURE_CANCEL_INFO_EXT};
        cancel_info.future = m_slowFooFuture;
        xrCancelFutureEXT(instance, &cancel_info);
        m_slowFooFuture = XR_NULL_FUTURE_EXT;
    }
}

void CheckFooRequestCompletion() {

    std::unique_lock<std::mutex> lock(m_mutex);
    // Check if a future is outstanding
    if (m_slowFooFuture == XR_NULL_FUTURE_EXT) {
        return;
    }

    // Poll for state of future
    XrFuturePollInfoEXT pollInfo{XR_TYPE_FUTURE_POLL_INFO_EXT};
    XrFuturePollResultEXT pollResult{XR_TYPE_FUTURE_POLL_RESULT_EXT};
    pollInfo.future = m_slowFooFuture;
    CHK_XR(xrPollFutureEXT(instance, &pollInfo, &pollResult));

    if (pollResult.state == XR_FUTURE_STATE_READY_EXT) {
        // Complete the future, consuming the result
        XrSlowFooCompletionEXT completion{XR_TYPE_SLOW_FOO_COMPLETION_EXT};
        XrResult result =
            xrSlowFooCompleteEXT(session, m_slowFooFuture, &completion);
        // Check XrResult from the completion function
        CHK_XR(result);
        // Check XrResult from the async operation
        CHK_XR(completion.futureResult);
        m_fooValue = completion.foo;
        m_slowFooFuture = XR_NULL_FUTURE_EXT;

        // Do something with the foo value.
    }
}

```

```

    }
}

void ThreadFunction() {
    while (!m_abort) {
        // other logic here

        CheckFooRequestCompletion();

        // sleep if needed.
    }
}

XrFutureEXT m_slowFooFuture{XR_NULL_FUTURE_EXT};
float m_fooValue{0.0f};
bool m_abort{false};
std::mutex m_mutex;
std::thread m_processThread;
};

```

## New Base Types

- [XrFutureEXT](#)

## New Functions

- [xrPollFutureEXT](#)
- [xrCancelFutureEXT](#)

## New Structures

- [XrFutureCompletionEXT](#)
- [XrFutureCompletionBaseHeaderEXT](#)
- [XrFuturePollInfoEXT](#)
- [XrFuturePollResultEXT](#)
- [XrFutureCancelInfoEXT](#)

## New Enum Constants

- [XR\\_NULL\\_FUTURE\\_EXT](#)

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_FUTURE\\_CANCEL\\_INFO\\_EXT](#)
- [XR\\_TYPE\\_FUTURE\\_POLL\\_INFO\\_EXT](#)

- `XR_TYPE_FUTURE_POLL_RESULT_EXT`
- `XR_TYPE_FUTURE_COMPLETION_EXT`

`XrResult` enumeration is extended with:

- `XR_ERROR_FUTURE_PENDING_EXT`
- `XR_ERROR_FUTURE_INVALID_EXT`

### Issues

- Should there be a state for completed functions that is separate from "invalid"?
  - Resolved.
  - Answer: No. This would force an implementing runtime to remember old futures forever. In order to allow implementations that delete all associated data about a future after completion, we cannot differentiate between a future that never existed and one that was completed. Similarly, invalidated/completed is not formally a "state" for futures in the final API.

### Version History

- Revision 1, 2023-02-14 (Andreas Løve Selvik, Meta Platforms and Ron Bessems, Magic Leap)
  - Initial extension description

## 12.30. XR\_EXT\_hand\_interaction

### Name String

`XR_EXT_hand_interaction`

### Extension Type

Instance extension

### Registered Extension Number

303

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Contributors

Yin Li, Microsoft

Alex Turner, Microsoft  
Casey Meekhof, Microsoft  
Lachlan Ford, Microsoft  
Eric Provencher, Unity Technologies  
Bryan Dube, Unity Technologies  
Peter Kuhn, Unity Technologies  
Tanya Li, Unity Technologies  
Jakob Bornecrantz, Collabora  
Jonathan Wright, Meta Platforms  
Federico Schliemann, Meta Platforms  
Andreas Loeve Selvik, Meta Platforms  
Nathan Nuber, Valve  
Joe Ludwig, Valve  
Rune Berg, Valve  
Adam Harwood, Ultraleap  
Robert Blenkinsopp, Ultraleap  
Paulo Gomes, Samsung Electronics  
Ron Bessems, Magic Leap  
Bastiaan Olij, Godot Engine

### 12.30.1. Overview

This extension defines four commonly used action poses for all user hand interaction profiles including both hand tracking devices and motion controller devices.

This extension also introduces a new interaction profile specifically designed for hand tracking devices to input through the [OpenXR action system](#). Though, for runtimes with controller inputs, the runtime **should** also provide this interaction profile through action mappings from the controller inputs, so that an application whose suggested action bindings solely depending on this hand interaction profile is usable on such runtimes as well.

### 12.30.2. Action poses for hand interactions

The following four action poses (i.e. "pinch," "poke," "aim," and "grip") enable a hand and finger interaction model, whether the tracking inputs are provided by a hand tracking device or a motion controller device.

The runtime **must** support all of the following action subpaths on all [interaction profiles](#) that are valid for the user paths of `/user/hand/left` and `/user/hand/right`, including those interaction profiles enabled through extensions.

- `.../input/aim/pose`
- `.../input/grip/pose`
- `.../input/pinch_ext/pose`

- `.../input/poke_ext/pose`

## Aim pose

The `.../input/aim/pose` is designed for interacting with objects out of arm's reach. For example, using a virtual laser pointer to aim at a virtual button on the wall is an interaction suited to the "aim" pose.

This is the same "aim" pose defined in [Standard pose identifiers](#). Every tracked controller profile already supports this pose.

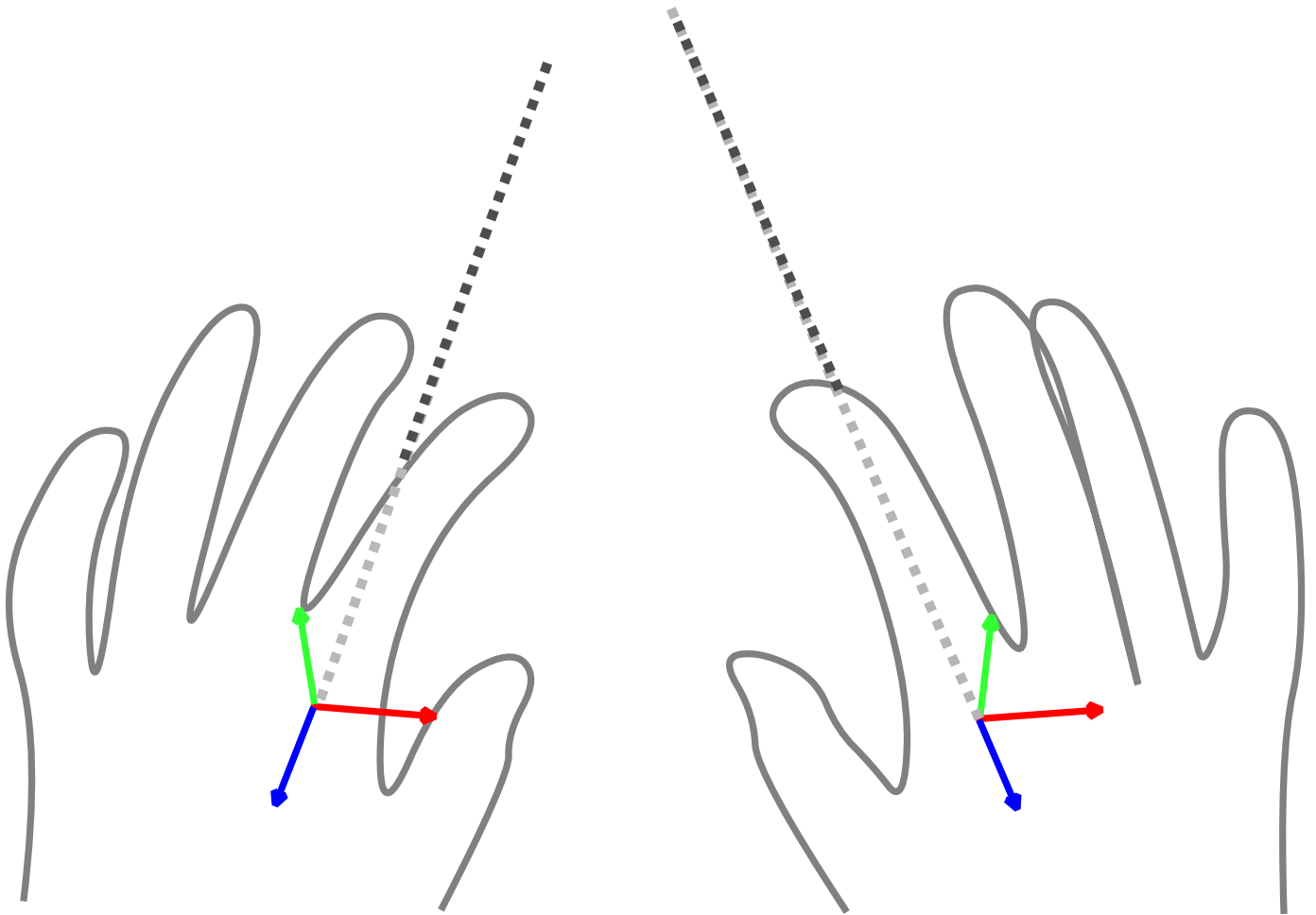


Figure 10. Example aim pose.

### Position

The position of an "aim" pose is typically in front of the user's hand and moves together with the corresponding hand, so that the user is able to easily see the aiming ray cast to the target in the world and adjust for aim.

### Orientation

The orientation of an "aim" pose is typically stabilized so that it is suitable to render an aiming ray emerging from the user's hand pointing into the world.

The -Z direction is the forward direction of the aiming gesture, that is, where the aiming ray is pointing at.

The +Y direction is a runtime defined direction based on the hand tracking device or ergonomics of the controller in the user's hand. It is typically pointing up in the world when the user is performing the aiming gesture naturally forward with a hand or controller in front of the user body.

The +X direction is orthogonal to +Y and +Z using the right-hand rule.

When targeting an object out of arm's reach, the runtime **may** optimize the "aim" pose stability for pointing at a target, therefore the rotation of the "aim" pose **may** account for forearm or shoulder motion as well as hand rotation. Hence, the "aim" pose **may** not always rigidly attach to the user's hand rotation. If the application desires to rotate the targeted remote object in place, it **should** use the rotation of the "grip" pose instead of "aim" pose, as if the user is remotely holding the object and rotating it.

### **Grip pose**

The `.../input/grip/pose` is designed for holding an object with a full hand grip gesture, for example, grasping and pushing a door's handle or holding and swinging a sword.

This is the same "grip" pose defined in [Standard pose identifiers](#). Every tracked controller profile already supports this pose.

The runtime **should** optimize the "grip" pose orientation so that it stabilizes large virtual objects held in the user's hand.



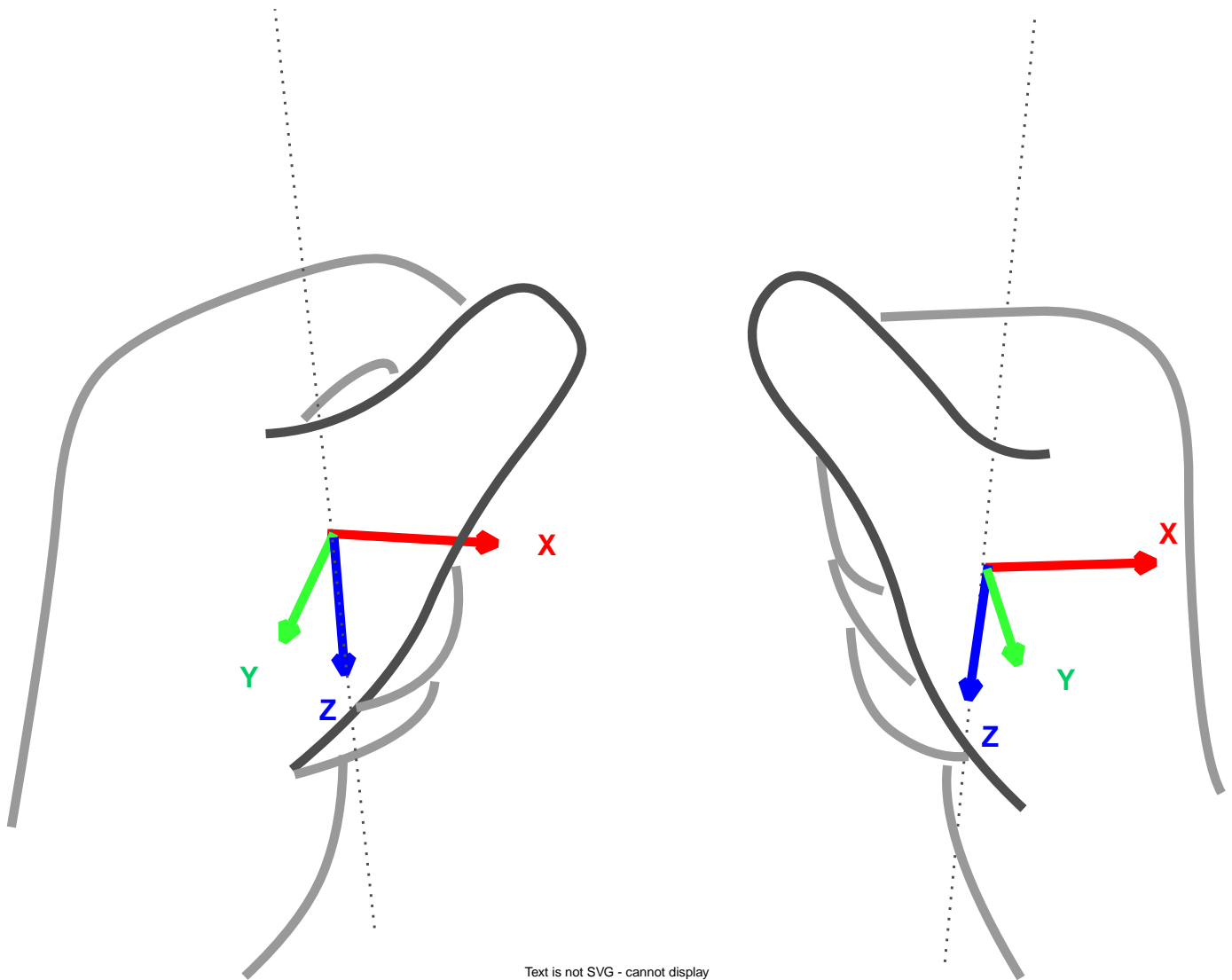


Figure 11. Example grip pose.

### Position

The position of the "grip" pose is at the centroid of the user's palm when the user makes a fist or holds a tube-like object in the hand.

### Orientation

The orientation of the "grip" pose **may** be used to render a virtual object held in the hand, for example, holding the grip of a virtual sword.

The Z axis of the grip pose goes through the center of the user's curled fingers when the user makes a fist or holds a controller, and the -Z direction (forward) goes from the little finger to the index finger.

When the user completely opens their hand to form a flat 5-finger pose and the palms face each other, the ray that is normal to the user's palms defines the X axis. The +X direction points away from the palm of the left hand and into the palm of the right hand. That is to say, in the described pose, the +X direction points to the user's right for both hands. To further illustrate: if the user is holding a stick by

making a fist with each hand in front of the body and pointing the stick up, the +X direction points to the user's right for both hands.

The +Y direction is orthogonal to +Z and +X using the right-hand rule.

## Pinch pose

The `.../input/pinch_ext/pose` is designed for interacting with a small object within arm's reach using a finger and thumb with a "pinch" gesture. For example, turning a key to open a lock or moving the knob on a slider control are interactions suited to the "pinch" pose.

The runtime **should** stabilize the "pinch" pose while the user is performing the "pinch" gesture.

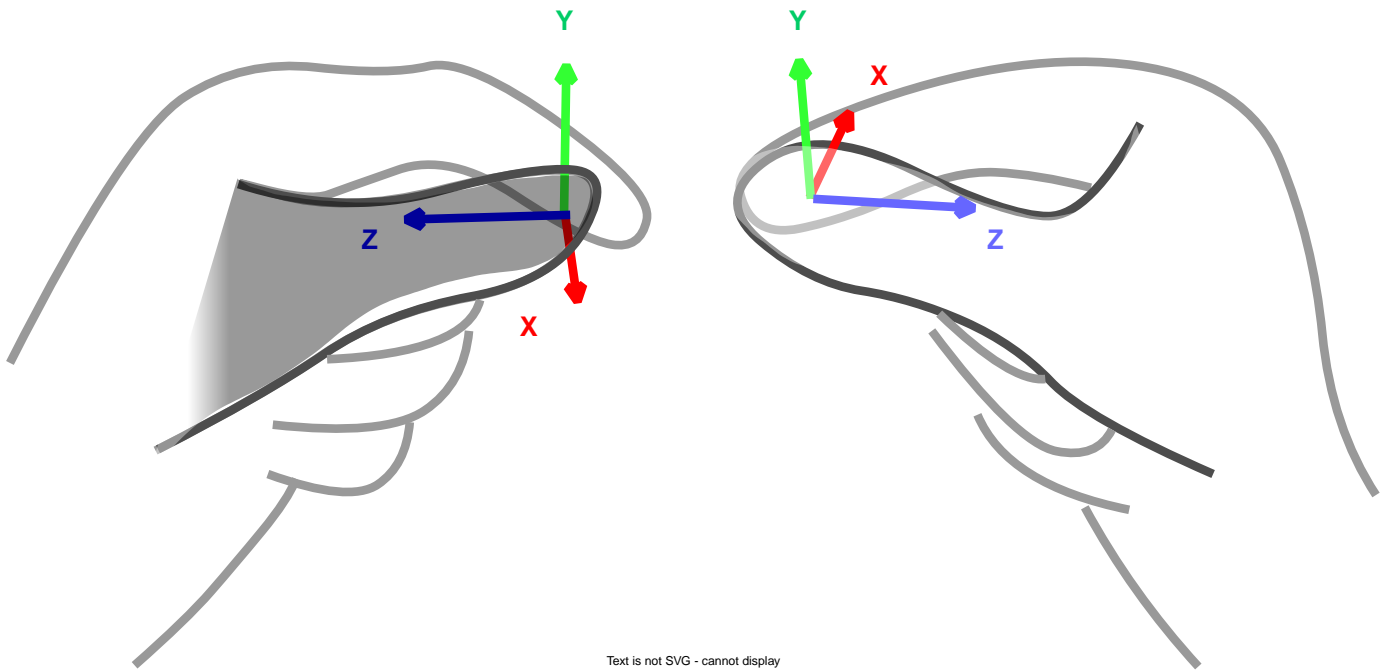


Figure 12. Example pinch pose.

## Position

When the input is provided by a hand tracking device, the position of the "pinch" pose is typically where the index and thumb fingertips will touch each other for a "pinch" gesture.

The runtime **may** provide the "pinch" pose using any finger based on the current user's preference for accessibility support. An application typically designs the "pinch" pose interaction assuming the "pinch" is performed using the index finger and thumb.

When the input is provided by a motion controller device, the position of the "pinch" pose is typically based on a fixed offset from the grip pose in front of the controller, where the user **can** naturally interact with a small object. The runtime **should** avoid obstructing the "pinch" pose with the physical profile of the motion controller.

## Orientation

The "pinch" pose orientation **must** rotate together with the hand rotation.

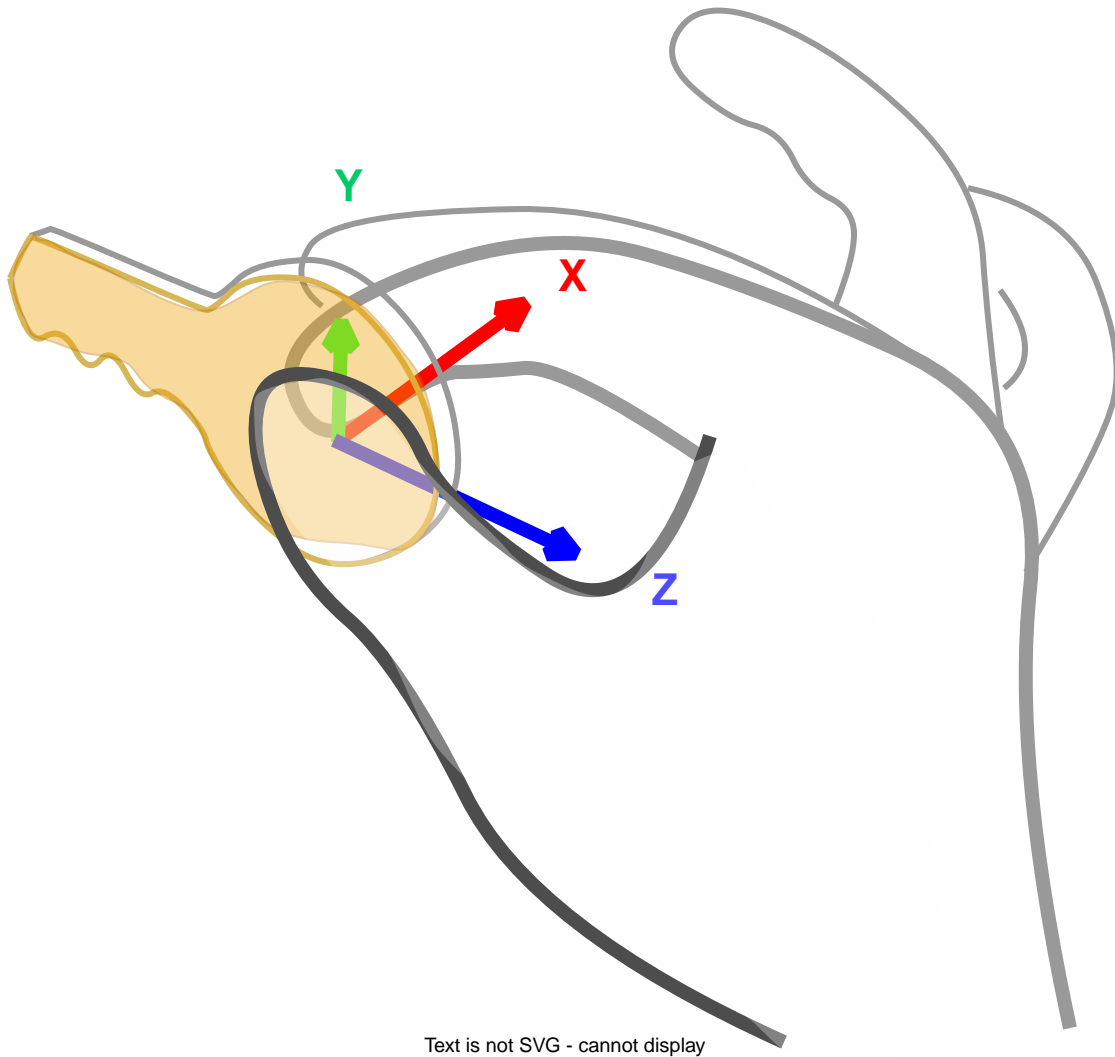


Figure 13. Example pinch orientation on right hand.

The "pinch" pose's orientation **may** be used to render a virtual object being held by a "pinch" gesture, for example, holding a key as illustrated in picture above.

If this virtual key is within a plane as illustrated in the above picture, the Y and Z axes of the "pinch" pose are within this plane.

The +Z axis is the backward direction of the "pinch" pose, typically the direction from the "pinch" position pointing to the mid point of thumb and finger proximal joints.

When the user puts both hands in front of the body at the same height, palms facing each other and fingers pointing forward, then performs a "pinch" gesture with both hands, the +Y direction for both hands **should** be roughly pointing up.

The X direction follows the right-hand rule using the Z and Y axes.

If the input is provided by a motion controller device, the orientation of the "pinch" pose is typically based on a fixed-rotation offset from the "grip" pose orientation that roughly follows the above

definition when the user is holding the controller naturally.

## Poke pose

The `.../input/poke_ext/pose` is designed for interactions using a fingertip to touch and push a small object. For example, pressing a push button with a fingertip, swiping to scroll a browser view, or typing on a virtual keyboard are interactions suited to the "poke" pose.

The application **may** use the "poke" pose as a point to interact with virtual objects, and this pose is typically enough for simple interactions.

The application **may** also use a volumetric representation of a "poke" gesture using a sphere combined with the "poke" pose. The center of such a sphere is located the distance of one radius in the +Z direction of the "poke" pose, such that the "poke" pose falls on the surface of the sphere and the sphere models the shape of the fingertip.

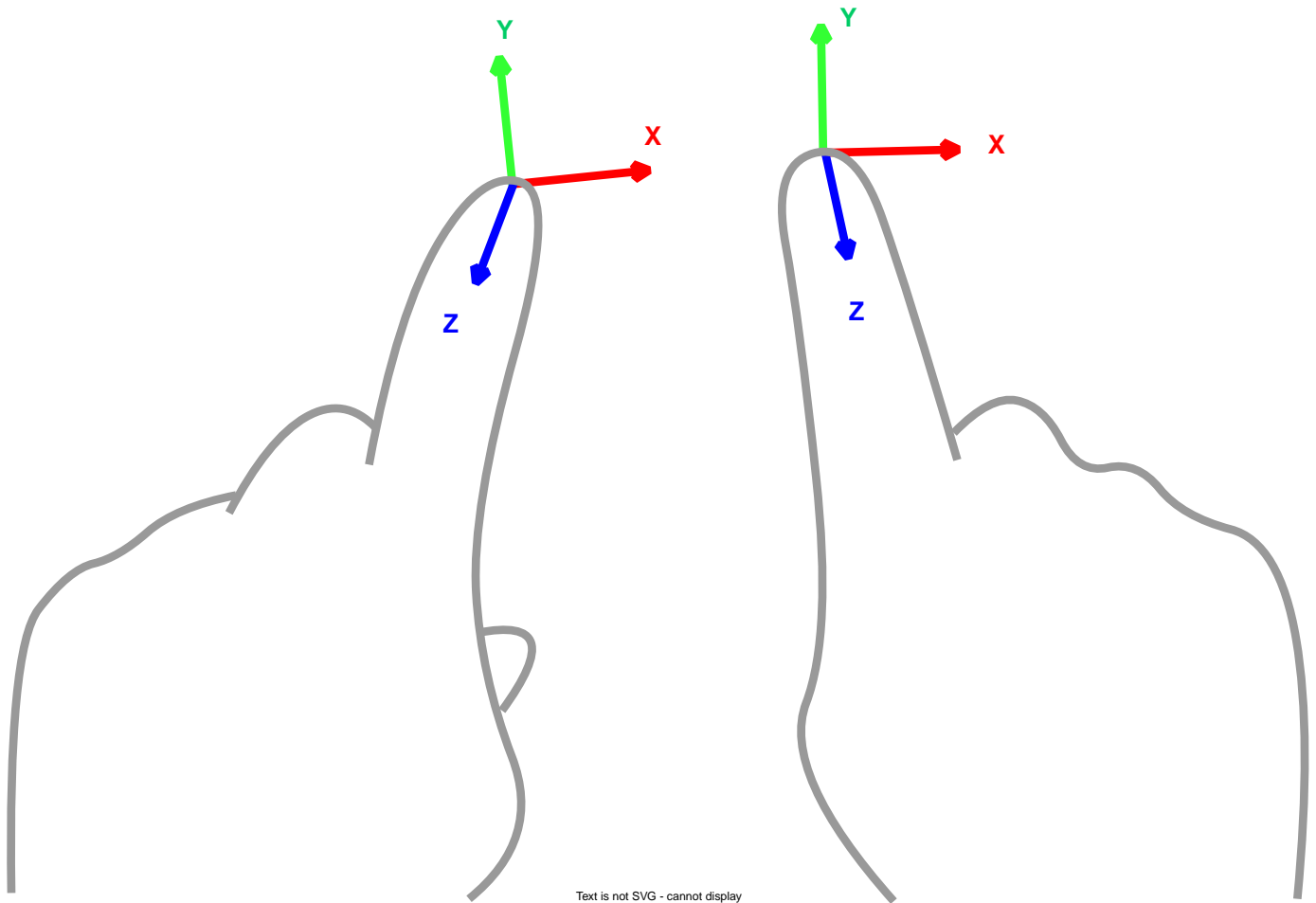


Figure 14. Example poke pose.

## Position

When input is provided by a hand tracking device, the position of the "poke" pose is at the surface of the extended index fingertip. The runtime **may** provide the "poke" pose using other fingers for accessibility support.

When input is provided by a motion controller, the position of the "poke" pose is typically based on a fixed offset from the "grip" pose in front of the controller, where touching and pushing a small object feels natural using the controller. The runtime **should** avoid obstructing the "poke" pose with the physical profile of the motion controller.

### Orientation

The +Y direction of the "poke" pose is the up direction in the world when the user is extending the index finger forward with palm facing down. When using a motion controller, +Y matches the up direction in the world when the user extends the index finger forward while holding the controller with palm facing down.

The +Z direction points from the fingertip towards the knuckle and parallel to the index finger distal bone, i.e. backwards when the user is holding a controller naturally in front of the body and pointing index finger forward.

The +X direction is orthogonal to +Y and +Z using the right-hand rule.

The "poke" pose **must** rotate together with the tip of the finger or the controller's "grip" pose.

### 12.30.3. The interaction profile for hand tracking devices

The hand interaction profile is designed for runtimes which provide hand inputs using hand tracking devices instead of controllers with triggers or buttons. This allows hand tracking devices to provide commonly used gestures and action poses to the [OpenXR action system](#).

In addition to hand tracking devices, runtimes with controller inputs **should** also implement this interaction profile through action bindings, so that an application whose suggested action bindings solely depending on this hand interaction profile is usable on such runtimes as well.

Interaction profile path:

- */interaction\_profiles/ext/hand\_interaction\_ext*

Valid for top level user path:

- */user/hand/left*
- */user/hand/right*

Supported component paths:

- *.../input/aim/pose*
- *.../input/grip/pose*
- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*
- *.../input/pinch\_ext/value*

- `.../input/pinch_ext/ready_ext`
- `.../input/aim_activate_ext/value`
- `.../input/aim_activate_ext/ready_ext`
- `.../input/grasp_ext/value`
- `.../input/grasp_ext/ready_ext`

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`

This interaction profile supports the above [four action poses](#), as well as the following three groups of action inputs.

### Pinch action

This interaction profile supports `.../input/pinch_ext/value` and `.../input/pinch_ext/ready_ext` actions.

The `.../input/pinch_ext/value` is a 1D analog input component indicating the extent which the user is bringing their finger and thumb together to perform a "pinch" gesture.

The `.../input/pinch_ext/value` **can** be used as either a boolean or float action type, where the value `XR_TRUE` or `1.0f` represents that the finger and thumb are touching each other.

The `.../input/pinch_ext/value` **must** be at value `0.0f` or `XR_FALSE` when the hand is in a natural and relaxed open state without the user making any extra effort.

The `.../input/pinch_ext/value` **should** be linear to the distance between the finger and thumb tips when they are in the range to change "pinch" value from 0 to 1.

The `.../input/pinch_ext/ready_ext` is a boolean input, where the value `XR_TRUE` indicates that the fingers

used to perform the "pinch" gesture are properly tracked by the hand tracking device and the hand shape is observed to be ready to perform or is performing a "pinch" gesture.

The `.../input/pinch_ext/value` **must** be `0.0f` or `XR_FALSE` when the `.../input/pinch_ext/ready_ext` is `XR_FALSE`.

The runtime **may** drive the input of the "pinch" gesture using any finger with the thumb to support accessibility.

### Aim activate action

This interaction profile supports `.../input/aim_activate_ext/value` and `.../input/aim_activate_ext/ready_ext` actions.

The `.../input/aim_activate_ext/value` is a 1D analog input component indicating that the user activated the action on the target that the user is pointing at with the aim pose.

The "aim\_activate" gesture is runtime defined, and it **should** be chosen so that the "aim" pose tracking is stable and usable for pointing at a distant target while the gesture is being performed.

The `.../input/aim_activate_ext/value` **can** be used as either a boolean or float action type, where the value `XR_TRUE` or `1.0f` represents that the aimed-at target is being fully interacted with.

The `.../input/aim_activate_ext/ready_ext` is a boolean input, where the value `XR_TRUE` indicates that the fingers to perform the "aim\_activate" gesture are properly tracked by the hand tracking device and the hand shape is observed to be ready to perform or is performing an "aim\_activate" gesture.

The `.../input/aim_activate_ext/value` **must** be `0.0f` or `XR_FALSE` when the `.../input/aim_activate_ext/ready_ext` is `XR_FALSE`.

### Grasp action

This interaction profile supports `.../input/grasp_ext/value` action.

The `.../input/grasp_ext/value` is a 1D analog input component indicating that the user is making a fist.

The `.../input/grasp_ext/value` **can** be used as either a boolean or float action type, where the value `XR_TRUE` or `1.0f` represents that the fist is tightly closed.

The `.../input/grasp_ext/value` **must** be at value `0.0f` or `XR_FALSE` when the hand is in a natural and relaxed open state without the user making any extra effort.

The `.../input/grasp_ext/ready_ext` is a boolean input, where the value `XR_TRUE` indicates that the hand performing the grasp action is properly tracked by the hand tracking device and it is observed to be ready to perform or is performing the grasp action.

The `.../input/grasp_ext/value` **must** be `0.0f` or `XR_FALSE` when the `.../input/grasp_ext/ready_ext` is `XR_FALSE`.

## Hand interaction gestures overlap

The values of the above "pinch", "grasp", and "aim\_activate" input actions **may** not be mutually exclusive when the input is provided by a hand tracking device. The application **should** not assume these actions are distinctively activated as action inputs provided by buttons or triggers on a controller. The application **should** suggest action bindings considering the intent of the action and their paired action pose.

## Using hand interaction profile with controllers

The runtimes with controller inputs **should** support the `/interaction_profiles/ext/hand_interaction_ext` profile using input mapping, so that applications **can** solely rely on the `/interaction_profiles/ext/hand_interaction_ext` profile to build XR experiences.

If the application desires to further customize the action poses with more flexible use of controller interaction profiles, the application **can** also provide action binding suggestions of controller profile using specific buttons or triggers to work together with the [commonly used four action poses](#).



## Typical usages of action poses with hand or controller profiles



- The `.../input/grip/pose` is typically used for holding a large object in the user's hand. When using a hand interaction profile, it is typically paired with `.../input/grasp_ext/value` for the user to directly manipulate an object held in a hand. When using a controller interaction profile, the "grip" pose is typically paired with a "squeeze" button or trigger that gives the user the sense of tightly holding an object.
- The `.../input/pinch_ext/pose` is typically used for directly manipulating a small object using the pinch gesture. When using a hand interaction profile, it is typically paired with the `.../input/pinch_ext/value` gesture. When using a controller interaction profile, it is typically paired with a trigger manipulated with the index finger, which typically requires curling the index finger and applying pressure with the fingertip.
- The `.../input/poke_ext/pose` is typically used for contact-based interactions using the motion of the hand or fingertip. It typically does not pair with other hand gestures or buttons on the controller. The application typically uses a sphere collider with the "poke" pose to visualize the pose and detect touch with a virtual object.
- The `.../input/aim/pose` is typically used for aiming at objects out of arm's reach. When using a hand interaction profile, it is typically paired with `.../input/aim_activate_ext/value` to optimize aiming ray stability while performing the gesture. When using a controller interaction profile, the "aim" pose is typically paired with a trigger or a button for aim and fire operations.
- Because controllers are typically mapping buttons or triggers for the above hand interaction values, they typically report `XR_TRUE` for their corresponding `.../ready_ext` action. This is because the buttons and triggers are always prepared and capable of receiving actions.

### New Object Types

### New Flag Types

### New Enum Constants

### New Enums

### New Structures

### New Functions

### Issues

### Version History

- Revision 1, 2021-08-06 (Yin Li)

- Initial extension description

## 12.31. XR\_EXT\_hand\_joints\_motion\_range

### Name String

`XR_EXT_hand_joints_motion_range`

### Extension Type

Instance extension

### Registered Extension Number

81

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_EXT\\_hand\\_tracking](#)

### Last Modified Date

2021-04-15

### IP Status

No known IP claims.

### Contributors

Joe van den Heuvel, Valve  
Rune Berg, Valve  
Joe Ludwig, Valve  
Jakob Bornecrantz, Collabora

### Overview

This extension augments the [XR\\_EXT\\_hand\\_tracking](#) extension to enable applications to request that the `XrHandJointLocationsEXT` returned by `xrLocateHandJointsEXT` should return hand joint locations conforming to a range of motion specified by the application.

The application **must** enable the [XR\\_EXT\\_hand\\_tracking](#) extension in order to use this extension.

### New Object Types

### New Flag Types

### New Enum Constants

## New Enums

The [XrHandJointsMotionRangeEXT](#) describes the hand joints' range of motion returned by [xrLocateHandJointsEXT](#).

Runtimes **must** support both [XR\\_HAND\\_JOINTS\\_MOTION\\_RANGE\\_CONFORMING\\_TO\\_CONTROLLER\\_EXT](#) and [XR\\_HAND\\_JOINTS\\_MOTION\\_RANGE\\_UNOBSTRUCTED\\_EXT](#) for each controller interaction profile that supports hand joint data.

```
// Provided by XR_EXT_hand_joints_motion_range
typedef enum XrHandJointsMotionRangeEXT {
    XR_HAND_JOINTS_MOTION_RANGE_UNOBSTRUCTED_EXT = 1,
    XR_HAND_JOINTS_MOTION_RANGE_CONFORMING_TO_CONTROLLER_EXT = 2,
    XR_HAND_JOINTS_MOTION_RANGE_MAX_ENUM_EXT = 0x7FFFFFFF
} XrHandJointsMotionRangeEXT;
```

### Enumerant Descriptions

- [XR\\_HAND\\_JOINTS\\_MOTION\\_RANGE\\_UNOBSTRUCTED\\_EXT](#) This option refers to the range of motion of a human hand, without any obstructions. Input systems that obstruct the movement of the user's hand (e.g.: a held controller preventing the user from making a fist) or have only limited ability to track finger positions **must** use the information available to them to emulate an unobstructed range of motion.
- [XR\\_HAND\\_JOINTS\\_MOTION\\_RANGE\\_CONFORMING\\_TO\\_CONTROLLER\\_EXT](#) This option refers to the range of motion of the hand joints taking into account any physical limits imposed by the controller itself. This will tend to be the most accurate pose compared to the user's actual hand pose, but might not allow a closed fist for example.
  - If the current interaction profile represents a controller, or other device that obstructs the hand, the implementation **must** return joint locations conforming to the shape of that device. If the current interaction profile is being emulated by a different physical controller, the implementation **may** return joint locations conforming to the shape of either the current interaction profile or the actual physical controller.
  - If the current interaction profile does not represent a controller, the implementation **must** return joint locations based on the unobstructed joint locations.

## New Structures

The [XrHandJointsMotionRangeInfoEXT](#) is a structure that an application **can** chain in [XrHandJointsLocateInfoEXT](#) to request the joint motion range specified by the [handJointsMotionRange](#) field.

Runtimes **must** return the appropriate joint locations depending on the `handJointsMotionRange` field and the currently active interaction profile.

```
// Provided by XR_EXT_hand_joints_motion_range
typedef struct XrHandJointsMotionRangeInfoEXT {
    XrStructureType          type;
    const void*              next;
    XrHandJointsMotionRangeEXT handJointsMotionRange;
} XrHandJointsMotionRangeInfoEXT;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `handJointsMotionRange` is an [XrHandJointsMotionRangeEXT](#) that defines the hand joint range of motion the application wants.

### Valid Usage (Implicit)

- The [XR\\_EXT\\_hand\\_joints\\_motion\\_range](#) extension **must** be enabled prior to using [XrHandJointsMotionRangeInfoEXT](#)
- `type` **must** be `XR_TYPE_HAND_JOINTS_MOTION_RANGE_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `handJointsMotionRange` **must** be a valid [XrHandJointsMotionRangeEXT](#) value

### New Functions

### Issues

### Version History

- Revision 1, 2021-04-15 (Rune Berg)
  - Initial extension description

## 12.32. XR\_EXT\_hand\_tracking

**Name String**

`XR_EXT_hand_tracking`

**Extension Type**

Instance extension

**Registered Extension Number**

52

**Revision**

4

**Extension and Version Dependencies**

[OpenXR 1.0](#)

**Last Modified Date**

2021-04-15

**IP Status**

No known IP claims.

**Contributors**

Yin Li, Microsoft

Lachlan Ford, Microsoft

Alex Turner, Microsoft

Bryce Hutchings, Microsoft

Cass Everitt, Oculus

Blake Taylor, Magic Leap

Joe van den Heuvel, Valve

Rune Berg, Valve

Valerie Benson, Ultraleap

Rylie Pavlik, Collabora

### 12.32.1. Overview

This extension 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.

### 12.32.2. Inspect system capability

An application **can** inspect whether the system is capable of hand tracking input by extending the [XrSystemProperties](#) with [XrSystemHandTrackingPropertiesEXT](#) structure when calling [xrGetSystemProperties](#).

```
// Provided by XR_EXT_hand_tracking
typedef struct XrSystemHandTrackingPropertiesEXT {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsHandTracking;
} XrSystemHandTrackingPropertiesEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsHandTracking` is an [XrBool32](#), indicating if current system is capable of hand tracking input.

## Valid Usage (Implicit)

- The [XR\\_EXT\\_hand\\_tracking](#) extension **must** be enabled prior to using [XrSystemHandTrackingPropertiesEXT](#)
- `type` **must** be `XR_TYPE_SYSTEM_HAND_TRACKING_PROPERTIES_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

If a runtime returns `XR_FALSE` for `supportsHandTracking`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateHandTrackerEXT](#).

### 12.32.3. Create a hand tracker handle

The [XrHandTrackerEXT](#) handle represents the resources for hand tracking of the specific hand.

```
XR_DEFINE_HANDLE(XrHandTrackerEXT)
```

An application creates separate [XrHandTrackerEXT](#) handles for left and right hands. This handle can be used to locate hand joints using [xrLocateHandJointsEXT](#) function.

A hand tracker provides joint locations with an unobstructed range of motion of an empty human hand.



### Note

This behavior can be modified by the [XR\\_EXT\\_hand\\_joints\\_motion\\_range](#) extension

An application can create an [XrHandTrackerEXT](#) handle using [xrCreateHandTrackerEXT](#) function.

```
// Provided by XR_EXT_hand_tracking
XrResult xrCreateHandTrackerEXT(
    XrSession session,
    const XrHandTrackerCreateInfoEXT* createInfo,
    XrHandTrackerEXT* handTracker);
```

### Parameter Descriptions

- **session** is an [XrSession](#) in which the hand tracker will be active.
- **createInfo** is the [XrHandTrackerCreateInfoEXT](#) used to specify the hand tracker.
- **handTracker** is the returned [XrHandTrackerEXT](#) handle.

### Valid Usage (Implicit)

- The [XR\\_EXT\\_hand\\_tracking](#) extension **must** be enabled prior to calling [xrCreateHandTrackerEXT](#)
- **session** **must** be a valid [XrSession](#) handle
- **createInfo** **must** be a pointer to a valid [XrHandTrackerCreateInfoEXT](#) structure
- **handTracker** **must** be a pointer to an [XrHandTrackerEXT](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

If the system does not support hand tracking, runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateHandTrackerEXT`. In this case, the runtime **must** return `XR_FALSE` for `XrSystemHandTrackingPropertiesEXT::supportsHandTracking` when the function `xrGetSystemProperties` is called, so that the application **can** avoid creating a hand tracker.

The `XrHandTrackerCreateInfoEXT` structure describes the information to create an `XrHandTrackerEXT` handle.

```
// Provided by XR_EXT_hand_tracking
typedef struct XrHandTrackerCreateInfoEXT {
    XrStructureType    type;
    const void*        next;
    XrHandEXT          hand;
    XrHandJointSetEXT  handJointSet;
} XrHandTrackerCreateInfoEXT;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `hand` is an [XrHandEXT](#) which describes which hand the tracker is tracking.
- `handJointSet` is an [XrHandJointSetEXT](#) describe the set of hand joints to retrieve.

## Valid Usage (Implicit)

- The [XR\\_EXT\\_hand\\_tracking](#) extension **must** be enabled prior to using [XrHandTrackerCreateInfoEXT](#)
- `type` **must** be `XR_TYPE_HAND_TRACKER_CREATE_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrHandPoseTypeInfoMSFT](#), [XrHandTrackingDataSourceInfoEXT](#)
- `hand` **must** be a valid [XrHandEXT](#) value
- `handJointSet` **must** be a valid [XrHandJointSetEXT](#) value

The [XrHandEXT](#) describes which hand the [XrHandTrackerEXT](#) is tracking.

```
// Provided by XR_EXT_hand_tracking
typedef enum XrHandEXT {
    XR_HAND_LEFT_EXT = 1,
    XR_HAND_RIGHT_EXT = 2,
    XR_HAND_MAX_ENUM_EXT = 0x7FFFFFFF
} XrHandEXT;
```

## Enumerant Descriptions

- `XR_HAND_LEFT_EXT` specifies the hand tracker will be tracking the user's left hand.
- `XR_HAND_RIGHT_EXT` specifies the hand tracker will be tracking the user's right hand.

The [XrHandJointSetEXT](#) enum describes the set of hand joints to track when creating an [XrHandTrackerEXT](#).

```
// Provided by XR_EXT_hand_tracking
typedef enum XrHandJointSetEXT {
    XR_HAND_JOINT_SET_DEFAULT_EXT = 0,
    // Provided by XR_ULTRALEAP_hand_tracking_forearm
    XR_HAND_JOINT_SET_HAND_WITH_FOREARM_ULTRALEAP = 1000149000,
    XR_HAND_JOINT_SET_MAX_ENUM_EXT = 0x7FFFFFFF
} XrHandJointSetEXT;
```

## Enumerant Descriptions

- `XR_HAND_JOINT_SET_DEFAULT_EXT` indicates that the created `XrHandTrackerEXT` tracks the set of hand joints described by `XrHandJointEXT` enum, i.e. the `xrLocateHandJointsEXT` function returns an array of joint locations with the count of `XR_HAND_JOINT_COUNT_EXT` and can be indexed using `XrHandJointEXT`.

`xrDestroyHandTrackerEXT` function releases the `handTracker` and the underlying resources when finished with hand tracking experiences.

```
// Provided by XR_EXT_hand_tracking
XrResult xrDestroyHandTrackerEXT(
    XrHandTrackerEXT          handTracker);
```

## Parameter Descriptions

- `handTracker` is an `XrHandTrackerEXT` previously created by `xrCreateHandTrackerEXT`.

## Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to calling `xrDestroyHandTrackerEXT`
- `handTracker` **must** be a valid `XrHandTrackerEXT` handle

## Thread Safety

- Access to `handTracker`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## 12.32.4. Locate hand joints

The `xrLocateHandJointsEXT` function locates an array of hand joints to a base space at given time.

```
// Provided by XR_EXT_hand_tracking
XrResult xrLocateHandJointsEXT(
    XrHandTrackerEXT          handTracker,
    const XrHandJointsLocateInfoEXT* locateInfo,
    XrHandJointLocationsEXT*  locations);
```

## Parameter Descriptions

- `handTracker` is an `XrHandTrackerEXT` previously created by `xrCreateHandTrackerEXT`.
- `locateInfo` is a pointer to `XrHandJointsLocateInfoEXT` describing information to locate hand joints.
- `locations` is a pointer to `XrHandJointLocationsEXT` receiving the returned hand joint locations.

## Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to calling `xrLocateHandJointsEXT`
- `handTracker` **must** be a valid `XrHandTrackerEXT` handle
- `locateInfo` **must** be a pointer to a valid `XrHandJointsLocateInfoEXT` structure
- `locations` **must** be a pointer to an `XrHandJointLocationsEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrHandJointsLocateInfoEXT` structure describes the information to locate hand joints.

```
// Provided by XR_EXT_hand_tracking
typedef struct XrHandJointsLocateInfoEXT {
    XrStructureType    type;
    const void*       next;
    XrSpace            baseSpace;
    XrTime             time;
} XrHandJointsLocateInfoEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `baseSpace` is an `XrSpace` within which the returned hand joint locations will be represented.
- `time` is an `XrTime` at which to locate the hand joints.

## Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to using `XrHandJointsLocateInfoEXT`
- `type` **must** be `XR_TYPE_HAND_JOINTS_LOCATE_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the next structure in a structure chain. See also: `XrHandJointsMotionRangeInfoEXT`
- `baseSpace` **must** be a valid `XrSpace` handle

`XrHandJointLocationsEXT` structure returns the state of the hand joint locations.

```
// Provided by XR_EXT_hand_tracking
typedef struct XrHandJointLocationsEXT {
    XrStructureType      type;
    void*                next;
    XrBool32             isActive;
    uint32_t             jointCount;
    XrHandJointLocationEXT* jointLocations;
} XrHandJointLocationsEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain, such as `XrHandJointVelocitiesEXT`.
- `isActive` is an `XrBool32` indicating if the hand tracker is actively tracking.
- `jointCount` is a `uint32_t` describing the count of elements in `jointLocations` array.
- `jointLocations` is an array of `XrHandJointLocationEXT` receiving the returned hand joint locations.

The application **must** allocate the memory for the output array `jointLocations` that can contain at least `jointCount` of `XrHandJointLocationEXT`.

The application **must** set `jointCount` as described by the `XrHandJointSetEXT` when creating the `XrHandTrackerEXT` otherwise the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

The runtime **must** return `jointLocations` representing the range of motion of a human hand, without any obstructions. Input systems that obstruct the movement of the user's hand (e.g.: a held controller

preventing the user from making a fist) or that have only limited ability to track finger positions **must** use the information available to them to emulate an unobstructed range of motion.

The runtime **must** update the `jointLocations` array ordered so that the application can index elements using the corresponding hand joint enum (e.g. `XrHandJointEXT`) as described by `XrHandJointSetEXT` when creating the `XrHandTrackerEXT`. For example, when the `XrHandTrackerEXT` is created with `XR_HAND_JOINT_SET_DEFAULT_EXT`, the application **must** set the `jointCount` to `XR_HAND_JOINT_COUNT_EXT`, and the runtime **must** fill the `jointLocations` array ordered so that it may be indexed by the `XrHandJointEXT` enum.

If the returned `isActive` is true, the runtime **must** return all joint locations with both `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` set. Although, in this case, some joint space locations **may** be untracked (i.e. `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` or `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` is unset).

If the returned `isActive` is false, it indicates the hand tracker did not detect the hand input or the application lost input focus. In this case, the runtime **must** return all `jointLocations` with neither `XR_SPACE_LOCATION_POSITION_VALID_BIT` nor `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` set.

### Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to using `XrHandJointLocationsEXT`
- `type` **must** be `XR_TYPE_HAND_JOINT_LOCATIONS_EXT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain. See also: `XrHandJointVelocitiesEXT`, `XrHandTrackingAimStateFB`, `XrHandTrackingCapsulesStateFB`, `XrHandTrackingDataSourceStateEXT`, `XrHandTrackingScaleFB`
- `jointLocations` **must** be a pointer to an array of `jointCount` `XrHandJointLocationEXT` structures
- The `jointCount` parameter **must** be greater than 0

`XrHandJointLocationEXT` structure describes the position, orientation, and radius of a hand joint.

```
// Provided by XR_EXT_hand_tracking
typedef struct XrHandJointLocationEXT {
    XrSpaceLocationFlags    locationFlags;
    XrPosef                 pose;
    float                   radius;
} XrHandJointLocationEXT;
```

## Member Descriptions

- `locationFlags` is a bitfield, with bit masks defined in `XrSpaceLocationFlagBits`, to indicate which members contain valid data. If none of the bits are set, no other fields in this structure **should** be considered to be valid or meaningful.
- `pose` is an `XrPosef` defining the position and orientation of the origin of a hand joint within the reference frame of the corresponding `XrHandJointsLocateInfoEXT::baseSpace`.
- `radius` is a `float` value radius of the corresponding joint in units of meters.

If the returned `locationFlags` has `XR_SPACE_LOCATION_POSITION_VALID_BIT` set, the returned radius **must** be a positive value.

If the returned `locationFlags` has `XR_SPACE_LOCATION_POSITION_VALID_BIT` unset, the returned radius value is undefined and should be avoided.

## Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to using `XrHandJointLocationEXT`
- `locationFlags` **must** be `0` or a valid combination of `XrSpaceLocationFlagBits` values

The application can chain an `XrHandJointVelocitiesEXT` structure to the `next` pointer of `XrHandJointLocationsEXT` when calling `xrLocateHandJointsEXT` to retrieve the hand joint velocities.

```
// Provided by XR_EXT_hand_tracking
typedef struct XrHandJointVelocitiesEXT {
    XrStructureType      type;
    void*                next;
    uint32_t             jointCount;
    XrHandJointVelocityEXT* jointVelocities;
} XrHandJointVelocitiesEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `jointCount` is a `uint32_t` describing the number of elements in `jointVelocities` array.
- `jointVelocities` is an array of `XrHandJointVelocityEXT` receiving the returned hand joint velocities.

The application **must** allocate the memory for the output array `jointVelocities` that can contain at least `jointCount` of `XrHandJointVelocityEXT`.

The application **must** input `jointCount` as described by the `XrHandJointSetEXT` when creating the `XrHandTrackerEXT`. Otherwise, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

The runtime **must** update the `jointVelocities` array in the order so that the application can index elements using the corresponding hand joint enum (e.g. `XrHandJointEXT`) as described by the `XrHandJointSetEXT` when creating the `XrHandTrackerEXT`. For example, when the `XrHandTrackerEXT` is created with `XR_HAND_JOINT_SET_DEFAULT_EXT`, the application **must** set the `jointCount` to `XR_HAND_JOINT_COUNT_EXT`, and the returned `jointVelocities` array **must** be ordered to be indexed by enum `XrHandJointEXT` enum.

If the returned `XrHandJointLocationsEXT::isActive` is false, it indicates the hand tracker did not detect a hand input or the application lost input focus. In this case, the runtime **must** return all `jointVelocities` with neither `XR_SPACE_VELOCITY_LINEAR_VALID_BIT` nor `XR_SPACE_VELOCITY_ANGULAR_VALID_BIT` set.

If an `XrHandJointVelocitiesEXT` structure is chained to `XrHandJointLocationsEXT::next`, the returned `XrHandJointLocationsEXT::isActive` is true, and the velocity is observed or can be calculated by the runtime, the runtime **must** fill in the linear velocity of each hand joint within the reference frame of `XrHandJointsLocateInfoEXT::baseSpace` and set the `XR_SPACE_VELOCITY_LINEAR_VALID_BIT`. Similarly, if an `XrHandJointVelocitiesEXT` structure is chained to `XrHandJointLocationsEXT::next`, the returned `XrHandJointLocationsEXT::isActive` is true, and the *angular velocity* is observed or can be calculated by the runtime, the runtime **must** fill in the angular velocity of each joint within the reference frame of `XrHandJointsLocateInfoEXT::baseSpace` and set the `XR_SPACE_VELOCITY_ANGULAR_VALID_BIT`.



## Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to using `XrHandJointVelocitiesEXT`
- `type` **must** be `XR_TYPE_HAND_JOINT_VELOCITIES_EXT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `jointVelocities` **must** be a pointer to an array of `jointCount` `XrHandJointVelocityEXT` structures
- The `jointCount` parameter **must** be greater than `0`

`XrHandJointVelocityEXT` structure describes the linear and angular velocity of a hand joint.

```
// Provided by XR_EXT_hand_tracking
typedef struct XrHandJointVelocityEXT {
    XrSpaceVelocityFlags    velocityFlags;
    XrVector3f              linearVelocity;
    XrVector3f              angularVelocity;
} XrHandJointVelocityEXT;
```

## Member Descriptions

- `velocityFlags` is a bitfield, with bit masks defined in `XrSpaceVelocityFlagBits`, to indicate which members contain valid data. If none of the bits are set, no other fields in this structure **should** be considered to be valid or meaningful.
- `linearVelocity` is the relative linear velocity of the hand joint with respect to and expressed in the reference frame of the corresponding `XrHandJointsLocateInfoEXT::baseSpace`, in units of meters per second.
- `angularVelocity` is the relative angular velocity of the hand joint with respect to the corresponding `XrHandJointsLocateInfoEXT::baseSpace`. The vector's direction is expressed in the reference frame of the corresponding `XrHandJointsLocateInfoEXT::baseSpace` and is parallel to the rotational axis of the hand joint. The vector's magnitude is the relative angular speed of the hand joint in radians per second. The vector follows the right-hand rule for torque/rotation.

## Valid Usage (Implicit)

- The `XR_EXT_hand_tracking` extension **must** be enabled prior to using `XrHandJointVelocityEXT`
- `velocityFlags` **must** be a valid combination of `XrSpaceVelocityFlagBits` values
- `velocityFlags` **must** not be 0

### 12.32.5. Example code for locating hand joints

The following example code demonstrates how to locate all hand joints relative to a world space.

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized
XrSpace worldSpace; // previously initialized, e.g. from
                    // XR_REFERENCE_SPACE_TYPE_LOCAL

// Inspect hand tracking system properties
XrSystemHandTrackingPropertiesEXT handTrackingSystemProperties{
    XR_TYPE_SYSTEM_HAND_TRACKING_PROPERTIES_EXT};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES,
                                   &handTrackingSystemProperties};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
if (!handTrackingSystemProperties.supportsHandTracking) {
    // The system does not support hand tracking
    return;
}

// Get function pointer for xrCreateHandTrackerEXT
PFN_xrCreateHandTrackerEXT pfnCreateHandTrackerEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateHandTrackerEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnCreateHandTrackerEXT)));

// Create a hand tracker for left hand that tracks default set of hand joints.
XrHandTrackerEXT leftHandTracker{};
{
    XrHandTrackerCreateInfoEXT createInfo{XR_TYPE_HAND_TRACKER_CREATE_INFO_EXT};
    createInfo.hand = XR_HAND_LEFT_EXT;
    createInfo.handJointSet = XR_HAND_JOINT_SET_DEFAULT_EXT;
    CHK_XR(pfnCreateHandTrackerEXT(session, &createInfo, &leftHandTracker));
}

// Allocate buffers to receive joint location and velocity data before frame
// loop starts
```

```

XrHandJointLocationEXT jointLocations[XR_HAND_JOINT_COUNT_EXT];
XrHandJointVelocityEXT jointVelocities[XR_HAND_JOINT_COUNT_EXT];

XrHandJointVelocitiesEXT velocities{XR_TYPE_HAND_JOINT_VELOCITIES_EXT};
velocities.jointCount = XR_HAND_JOINT_COUNT_EXT;
velocities.jointVelocities = jointVelocities;

XrHandJointLocationsEXT locations{XR_TYPE_HAND_JOINT_LOCATIONS_EXT};
locations.next = &velocities;
locations.jointCount = XR_HAND_JOINT_COUNT_EXT;
locations.jointLocations = jointLocations;

// Get function pointer for xrLocateHandJointsEXT
PFN_xrLocateHandJointsEXT pfnLocateHandJointsEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrLocateHandJointsEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                             &pfnLocateHandJointsEXT)));

while (1) {
    // ...
    // For every frame in frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrHandJointsLocateInfoEXT locateInfo{XR_TYPE_HAND_JOINTS_LOCATE_INFO_EXT};
    locateInfo.baseSpace = worldSpace;
    locateInfo.time = time;

    CHK_XR(pfnLocateHandJointsEXT(leftHandTracker, &locateInfo, &locations));

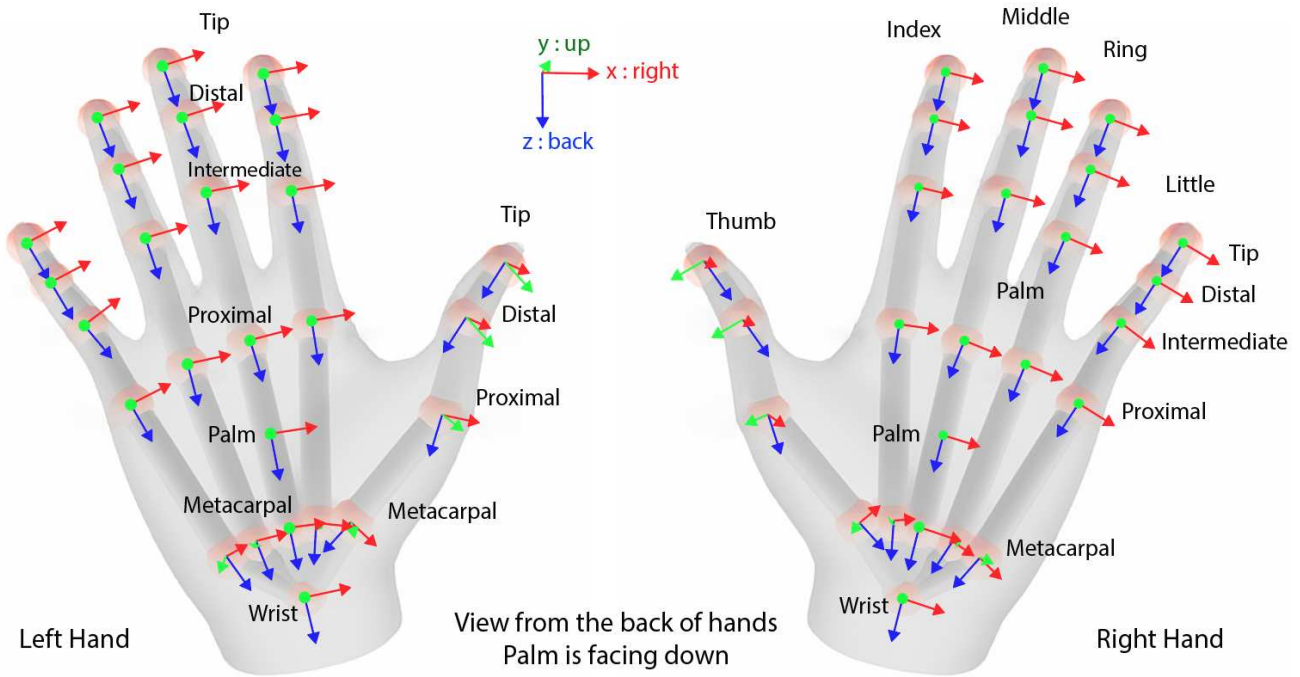
    if (locations.isActive) {
        // The returned joint location array can be directly indexed with
        // XrHandJointEXT enum.
        const XrPosef &indexTipInWorld =
            jointLocations[XR_HAND_JOINT_INDEX_TIP_EXT].pose;
        const XrPosef &thumbTipInWorld =
            jointLocations[XR_HAND_JOINT_THUMB_TIP_EXT].pose;

        // using the returned radius and velocity of index finger tip.
        const float indexTipRadius =
            jointLocations[XR_HAND_JOINT_INDEX_TIP_EXT].radius;
        const XrHandJointVelocityEXT &indexTipVelocity =
            jointVelocities[XR_HAND_JOINT_INDEX_TIP_EXT];
    }
}

```

## 12.32.6. Conventions of hand joints

This extension defines 26 joints for hand tracking: 4 joints for the thumb finger, 5 joints for the other four fingers, and the wrist and palm of the hands.



```
// Provided by XR_EXT_hand_tracking
typedef enum XrHandJointEXT {
    XR_HAND_JOINT_PALM_EXT = 0,
    XR_HAND_JOINT_WRIST_EXT = 1,
    XR_HAND_JOINT_THUMB_METACARPAL_EXT = 2,
    XR_HAND_JOINT_THUMB_PROXIMAL_EXT = 3,
    XR_HAND_JOINT_THUMB_DISTAL_EXT = 4,
    XR_HAND_JOINT_THUMB_TIP_EXT = 5,
    XR_HAND_JOINT_INDEX_METACARPAL_EXT = 6,
    XR_HAND_JOINT_INDEX_PROXIMAL_EXT = 7,
    XR_HAND_JOINT_INDEX_INTERMEDIATE_EXT = 8,
    XR_HAND_JOINT_INDEX_DISTAL_EXT = 9,
    XR_HAND_JOINT_INDEX_TIP_EXT = 10,
    XR_HAND_JOINT_MIDDLE_METACARPAL_EXT = 11,
    XR_HAND_JOINT_MIDDLE_PROXIMAL_EXT = 12,
    XR_HAND_JOINT_MIDDLE_INTERMEDIATE_EXT = 13,
    XR_HAND_JOINT_MIDDLE_DISTAL_EXT = 14,
    XR_HAND_JOINT_MIDDLE_TIP_EXT = 15,
    XR_HAND_JOINT_RING_METACARPAL_EXT = 16,
    XR_HAND_JOINT_RING_PROXIMAL_EXT = 17,
    XR_HAND_JOINT_RING_INTERMEDIATE_EXT = 18,
    XR_HAND_JOINT_RING_DISTAL_EXT = 19,
    XR_HAND_JOINT_RING_TIP_EXT = 20,
    XR_HAND_JOINT_LITTLE_METACARPAL_EXT = 21,
    XR_HAND_JOINT_LITTLE_PROXIMAL_EXT = 22,
    XR_HAND_JOINT_LITTLE_INTERMEDIATE_EXT = 23,
    XR_HAND_JOINT_LITTLE_DISTAL_EXT = 24,
    XR_HAND_JOINT_LITTLE_TIP_EXT = 25,
    XR_HAND_JOINT_MAX_ENUM_EXT = 0x7FFFFFFF
} XrHandJointEXT;
```

The finger joints, except the tips, are named after the corresponding bone at the further end of the bone from the finger tips. The joint's orientation is defined at a fully opened hand pose facing down as in the above picture.

#### Note



Many applications and game engines use names to identify joints rather than using indices. If possible, applications should use the joint name part of the `XrHandJointEXT` enum plus a hand identifier to help prevent joint name clashes (e.g. `Index_Metacarpal_L`, `Thumb_Tip_R`). Using consistent names increases the portability of assets between applications and engines. Including the hand in the identifier prevents ambiguity when both hands are used in the same skeleton, such as when they are combined with additional joints to form a full body skeleton.

The backward (+Z) direction is parallel to the corresponding bone and points away from the finger tip.

The up (+Y) direction is pointing out of the back of and perpendicular to the corresponding finger nail at the fully opened hand pose. The X direction is perpendicular to Y and Z and follows the right hand rule.

The wrist joint is located at the pivot point of the wrist which is location invariant when twisting hand without moving the forearm. The backward (+Z) direction is parallel to the line from wrist joint to middle finger metacarpal joint, and points away from the finger tips. The up (+Y) direction points out towards back of hand and perpendicular to the skin at wrist. The X direction is perpendicular to the Y and Z directions and follows the right hand rule.

The palm joint is located at the center of the middle finger's metacarpal bone. The backward (+Z) direction is parallel to the middle finger's metacarpal bone, and points away from the finger tips. The up (+Y) direction is perpendicular to palm surface and pointing towards the back of the hand. The X direction is perpendicular to the Y and Z directions and follows the right hand rule.

The radius of each joint is the distance from the joint to the skin in meters. The application can use a sphere at the joint location with joint radius for collision detection for interactions, such as pushing a virtual button using the index finger tip.

For example, suppose the radius of the palm joint is  $r$  then the app **can** offset  $\{0, -r, 0\}$  to palm joint location to get the surface of hand palm center, or offset  $\{0, r, 0\}$  to get the back surface of the hand.

Note that the palm joint for the hand tracking is not the same as *.../input/grip/pose* when hand tracking is provided by controller tracking. A "grip" pose is located at the center of the controller handle when user is holding a controller, outside of the user's hand. A "palm" pose is located at the center of middle finger metacarpal bone which is inside the user's hand.

```
// Provided by XR_EXT_hand_tracking
#define XR_HAND_JOINT_COUNT_EXT 26
```

[XR\\_HAND\\_JOINT\\_COUNT\\_EXT](#) defines the number of hand joint enumerants defined in [XrHandJointEXT](#)

### New Object Types

- [XrHandTrackerEXT](#)

### New Flag Types

### New Enum Constants

- [XR\\_HAND\\_JOINT\\_COUNT\\_EXT](#)

[XrObjectType](#) enumeration is extended with:

- `XR_OBJECT_TYPE_HAND_TRACKER_EXT`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SYSTEM_HAND_TRACKING_PROPERTIES_EXT`
- `XR_TYPE_HAND_TRACKER_CREATE_INFO_EXT`
- `XR_TYPE_HAND_JOINTS_LOCATE_INFO_EXT`
- `XR_TYPE_HAND_JOINT_LOCATIONS_EXT`
- `XR_TYPE_HAND_JOINT_VELOCITIES_EXT`

### New Enums

- `XrHandEXT`
- `XrHandJointEXT`
- `XrHandJointSetEXT`

### New Structures

- `XrSystemHandTrackingPropertiesEXT`
- `XrHandTrackerCreateInfoEXT`
- `XrHandJointsLocateInfoEXT`
- `XrHandJointLocationEXT`
- `XrHandJointVelocityEXT`
- `XrHandJointLocationsEXT`
- `XrHandJointVelocitiesEXT`

### New Functions

- `xrCreateHandTrackerEXT`
- `xrDestroyHandTrackerEXT`
- `xrLocateHandJointsEXT`

### Issues

### Version History

- Revision 1, 2019-09-16 (Yin LI)
  - Initial extension description
- Revision 2, 2020-04-20 (Yin LI)
  - Replace hand joint spaces to locate hand joints function.

- Revision 3, 2021-04-13 (Rylie Pavlik, Rune Berg)
  - Fix example code to properly use `xrGetInstanceProcAddr`.
  - Add recommended bone names
- Revision 4, 2021-04-15 (Rune Berg)
  - Clarify that use of this extension produces an unobstructed hand range of motion.

## 12.33. XR\_EXT\_hand\_tracking\_data\_source

### Name String

`XR_EXT_hand_tracking_data_source`

### Extension Type

Instance extension

### Registered Extension Number

429

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_EXT\\_hand\\_tracking](#)

### Last Modified Date

2023-01-23

### IP Status

No known IP claims.

### Contributors

Jakob Bornecrantz, Collabora  
John Kearney, Meta  
Robert Memmott, Meta  
Andreas Selvik, Meta  
Yin Li, Microsoft  
Robert Blenkinsopp, Ultraleap  
Nathan Nuber, Valve

### Contacts

John Kearney, Meta



## Overview

This extension augments the [XR\\_EXT\\_hand\\_tracking](#) extension.

Runtimes **may** support a variety of data sources for hand joint data for [XR\\_EXT\\_hand\\_tracking](#), and some runtimes and devices **may** use joint data from multiple sources. This extension allows an application and the runtime to communicate about and make use of those data sources in a cooperative manner.

This extension allows the application to specify the data sources that it wants data from when creating a hand tracking handle, and allows the runtime to specify the currently active data source.

The application **must** enable the [XR\\_EXT\\_hand\\_tracking](#) extension in order to use this extension.

The [XrHandTrackingDataSourceEXT](#) enum describes a hand tracking data source when creating an [XrHandTrackerEXT](#) handle.

```
// Provided by XR_EXT_hand_tracking_data_source
typedef enum XrHandTrackingDataSourceEXT {
    XR_HAND_TRACKING_DATA_SOURCE_UNOBSTRUCTED_EXT = 1,
    XR_HAND_TRACKING_DATA_SOURCE_CONTROLLER_EXT = 2,
    XR_HAND_TRACKING_DATA_SOURCE_MAX_ENUM_EXT = 0x7FFFFFFF
} XrHandTrackingDataSourceEXT;
```

The application **can** use [XrHandTrackingDataSourceEXT](#) with [XrHandTrackingDataSourceInfoEXT](#) when calling [xrCreateHandTrackerEXT](#) to tell the runtime all supported data sources for the application for the hand tracking inputs.

The application **can** use it with [XrHandTrackingDataSourceStateEXT](#) when calling [xrLocateHandJointsEXT](#) to inspect what data source the runtime used for the returned hand joint locations.

If the [XR\\_EXT\\_hand\\_joints\\_motion\\_range](#) extension is supported by the runtime and the data source is [XR\\_HAND\\_TRACKING\\_DATA\\_SOURCE\\_CONTROLLER\\_EXT](#), then it is expected that application will use that extension when retrieving hand joint poses.

## Enumerant Descriptions

- `XR_HAND_TRACKING_DATA_SOURCE_UNOBSERVED_EXT` - This data source value indicates that the hand tracking data source supports using individual fingers and joints separately. Examples of such sources include optical hand tracking, data gloves, or motion capture devices.
- `XR_HAND_TRACKING_DATA_SOURCE_CONTROLLER_EXT` - This data source value indicates that the hand tracking data source is a motion controller. The runtime **must** not supply this data source if the controller providing the data is not actively held in the user's hand, but **may** still provide data if the runtime is unable to detect if the controller is not in the user's hand, or a user selected policy changes this behavior. Unless specified otherwise by another extension, data returned from `XR_HAND_TRACKING_DATA_SOURCE_CONTROLLER_EXT` **must** behave as `XR_HAND_JOINTS_MOTION_RANGE_UNOBSERVED_EXT`.

The `XrHandTrackingDataSourceInfoEXT` structure is defined as:

```
// Provided by XR_EXT_hand_tracking_data_source
typedef struct XrHandTrackingDataSourceInfoEXT {
    XrStructureType          type;
    const void*              next;
    uint32_t                  requestedDataSourceCount;
    XrHandTrackingDataSourceEXT* requestedDataSources;
} XrHandTrackingDataSourceInfoEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `requestedDataSourceCount` is the number of elements in the `requestedDataSources` array.
- `requestedDataSources` is an array of `XrHandTrackingDataSourceEXT` that the application accepts.

The `XrHandTrackingDataSourceInfoEXT` is a structure that an application **can** chain to `XrHandTrackerCreateInfoEXT::next` to specify the hand tracking data sources that the application accepts.

Because the hand tracking device **may** change during a running session, the runtime **may** return a valid `XrHandTrackerEXT` handle even if there is no currently active hand tracking device or the active device does not satisfy any or all data sources requested by the application's call to `xrCreateHandTrackerEXT`. The runtime **may** instead return `XR_ERROR_FEATURE_UNSUPPORTED` from

`xrCreateHandTrackerEXT`, if for example the runtime believes it will never be able to satisfy the request.

If any value in `requestedDataSources` is duplicated, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE` from the call to `xrCreateHandTrackerEXT`. If `requestedDataSourceCount` is 0, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE` from the call to `xrCreateHandTrackerEXT`.

### Valid Usage (Implicit)

- The `XR_EXT_hand_tracking_data_source` extension **must** be enabled prior to using `XrHandTrackingDataSourceInfoEXT`
- `type` **must** be `XR_TYPE_HAND_TRACKING_DATA_SOURCE_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `requestedDataSourceCount` is not 0, `requestedDataSources` **must** be a pointer to an array of `requestedDataSourceCount` `XrHandTrackingDataSourceEXT` values

The `XrHandTrackingDataSourceStateEXT` structure is defined as:

```
// Provided by XR_EXT_hand_tracking_data_source
typedef struct XrHandTrackingDataSourceStateEXT {
    XrStructureType          type;
    void*                    next;
    XrBool32                 isActive;
    XrHandTrackingDataSourceEXT dataSource;
} XrHandTrackingDataSourceStateEXT;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `isActive` indicating there is an active data source
- `dataSource` indicating the data source that was used to generate the hand tracking joints.

`XrHandTrackingDataSourceStateEXT` is a structure that an application **can** chain to `XrHandJointLocationsEXT::next` when calling `xrLocateHandJointsEXT` to retrieve the data source of the currently active hand tracking device.

When the returned `isActive` is `XR_FALSE`, it indicates the currently active hand tracking device does not

support any of the requested data sources. In these cases, the runtime **must** also return no valid tracking locations for hand joints from this [xrLocateHandJointsEXT](#) function.

If the tracker was not created with [XrHandTrackingDataSourceInfoEXT](#) chained to [XrHandTrackerCreateInfoEXT::next](#), then the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`, if [XrHandTrackingDataSourceStateEXT](#) is passed in the call to [xrLocateHandJointsEXT](#).

If there is an active hand tracking device that is one of the specified [XrHandTrackingDataSourceInfoEXT::requestedDataSources](#), the runtime **must** set `isActive` to `XR_TRUE`. When the runtime sets `isActive` to `XR_TRUE`, the runtime **must** set `dataSource` indicate the active data source. The runtime **must** return a `dataSource` that is a subset of the [XrHandTrackingDataSourceInfoEXT::requestedDataSources](#) when creating the corresponding hand tracker.

### Valid Usage (Implicit)

- The `XR_EXT_hand_tracking_data_source` extension **must** be enabled prior to using [XrHandTrackingDataSourceStateEXT](#)
- `type` **must** be `XR_TYPE_HAND_TRACKING_DATA_SOURCE_STATE_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `dataSource` **must** be a valid [XrHandTrackingDataSourceEXT](#) value

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with: \* `XR_TYPE_HAND_TRACKING_DATA_SOURCE_INFO_EXT` \* `XR_TYPE_HAND_TRACKING_DATA_SOURCE_STATE_EXT`

## New Enums

- [XrHandTrackingDataSourceEXT](#)

## New Structures

- [XrHandTrackingDataSourceInfoEXT](#)
- [XrHandTrackingDataSourceStateEXT](#)

## New Functions

## Issues

1. Should this extension require `XR_HAND_JOINTS_MOTION_RANGE_CONFORMING_TO_CONTROLLER_EXT` if the

data source is `XR_HAND_TRACKING_DATA_SOURCE_CONTROLLER_EXT` and `XR_EXT_hand_joints_motion_range` is not enabled?

**RESOLVED:** Yes.

It should not be required. We expect that a key use of the data from this extension will be replicating data hand tracking joint data for social purposes. For that use-case, the data returned in the style of `XR_HAND_JOINTS_MOTION_RANGE_UNOBSTRUCTED_EXT` is more appropriate.

This is consistent with `XR_EXT_hand_tracking` extension which requires that the `jointLocations` represent the range of motion of a human hand, without any obstructions.

2. Should `XrHandTrackingDataSourceInfoEXT` include an `isActive` member or can it use `isActive` from `XrHandJointLocationsEXT`?

**RESOLVED:** Yes.

Yes; `XrHandTrackingDataSourceInfoEXT` needs to include the `isActive` member and cannot use the `isActive` from `XrHandJointLocationsEXT` as the meaning of these members is different.

The `isActive` member of `XrHandTrackingDataSourceStateEXT` allows the runtime to describe if the tracking device is active. `XrHandTrackingDataSourceStateEXT::isActive` describes if the tracking device is actively tracking. It is possible for a data source to be active but not actively tracking and we want to represent if the device is active in this extension.

## Version History

- Revision 1, 2023-01-23 (John Kearney)
  - Initial extension description

## 12.34. XR\_EXT\_performance\_settings

### Name String

`XR_EXT_performance_settings`

### Extension Type

Instance extension

### Registered Extension Number

16

### Revision

4

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-04-14

## IP Status

No known IP claims.

## Contributors

Armelle Laine, Qualcomm Technologies Inc, on behalf of Qualcomm Innovation Center, Inc  
Rylie Pavlik, Collabora

### 12.34.1. Overview

This extension defines an API for the application to give performance hints to the runtime and for the runtime to send performance related notifications back to the application. This allows both sides to dial in a suitable compromise between needed CPU and GPU performance, thermal sustainability and a consistent good user experience throughout the session.

The goal is to render frames consistently, in time, under varying system load without consuming more energy than necessary.

In summary, the APIs allow:

- setting performance level hints
- receiving performance related notifications

### 12.34.2. Setting Performance Levels Hints

#### Performance level hint definition

The XR performance level hints for a given hardware system are expressed as a level [XrPerfSettingsLevelEXT](#) for each of the XR-critical processing domains [XrPerfSettingsDomainEXT](#) (currently defined is a CPU and a GPU domain):

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsDomainEXT {
    XR_PERF_SETTINGS_DOMAIN_CPU_EXT = 1,
    XR_PERF_SETTINGS_DOMAIN_GPU_EXT = 2,
    XR_PERF_SETTINGS_DOMAIN_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsDomainEXT;
```

```
// Provided by XR_EXT_performance_settings
typedef enum XrPerfSettingsLevelEXT {
    XR_PERF_SETTINGS_LEVEL_POWER_SAVINGS_EXT = 0,
    XR_PERF_SETTINGS_LEVEL_SUSTAINED_LOW_EXT = 25,
    XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT = 50,
    XR_PERF_SETTINGS_LEVEL_BOOST_EXT = 75,
    XR_PERF_SETTINGS_LEVEL_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsLevelEXT;
```

This extension defines platform-independent level hints:

- `XR_PERF_SETTINGS_LEVEL_POWER_SAVINGS_EXT` is used by the application to indicate that it enters a non-XR section (head-locked / static screen), during which power savings are to be prioritized. Consistent XR compositing, consistent frame rendering, and low latency are not needed.
- `XR_PERF_SETTINGS_LEVEL_SUSTAINED_LOW_EXT` is used by the application to indicate that it enters a low and stable complexity section, during which reducing power is more important than occasional late rendering frames. With such a hint, the XR Runtime still strives for consistent XR compositing (no tearing) within a thermally sustainable range(\*), but is allowed to take measures to reduce power, such as increasing latencies or reducing headroom.
- `XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT` is used by the application to indicate that it enters a high or dynamic complexity section, during which the XR Runtime strives for consistent XR compositing and frame rendering within a thermally sustainable range(\*).
- `XR_PERF_SETTINGS_LEVEL_BOOST_EXT` is used to indicate that the application enters a section with very high complexity, during which the XR Runtime is allowed to step up beyond the thermally sustainable range. As not thermally sustainable, this level is meant to be used for short-term durations (< 30 seconds).

(\*) If the application chooses one of the two sustainable levels (`XR_PERF_SETTINGS_LEVEL_SUSTAINED_LOW_EXT` or `XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT`), the device may still run into thermal limits under non-nominal circumstances (high room temperature, additional background loads, extended device operation) and therefore the application should also in the sustainable modes be prepared to react to performance notifications (in particular `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` and `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` in the thermal sub-domain, see [Notification level definition](#)).

The XR Runtime shall select `XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT` as the default hint if the application does not provide any. The function to call for setting performance level hints is [xrPerfSettingsSetPerformanceLevelEXT](#).

```
// Provided by XR_EXT_performance_settings
XrResult xrPerfSettingsSetPerformanceLevelEXT(
    XrSession session,
    XrPerfSettingsDomainEXT domain,
    XrPerfSettingsLevelEXT level);
```

### Example of using the short-term boost level hint

For a limited amount of time, both the Mobile and PC systems can provide a higher level of performance than is thermally sustainable. It is desirable to make this extra computational power available for short complex scenes, then go back to a sustainable lower level. This section describes means for the application developer to apply settings directing the runtime to boost performance for a short-term duration.

The application developer must pay attention to keep these boost periods very short and carefully monitor the side effects, which may vary a lot between different hardware systems.

#### Sample code for temporarily boosting the performance

```
1 extern XrInstance instance; ①
2 extern XrSession session;
3
4 // Get function pointer for xrPerfSettingsSetPerformanceLevelEXT
5 PFN_xrPerfSettingsSetPerformanceLevelEXT pfnPerfSettingsSetPerformanceLevelEXT;
6 CHK_XR(xrGetInstanceProcAddr(instance, "xrPerfSettingsSetPerformanceLevelEXT",
7     (PFN_xrVoidFunction*)(
8     &pfnPerfSettingsSetPerformanceLevelEXT)));
9
10 // before entering the high complexity section
11 pfnPerfSettingsSetPerformanceLevelEXT(session, XR_PERF_SETTINGS_DOMAIN_CPU_EXT,
12     XR_PERF_SETTINGS_LEVEL_BOOST_EXT); ②
13
14 // entering the high complexity section
15 // ... running
16 // end of the high complexity section
17
18 pfnPerfSettingsSetPerformanceLevelEXT(session, XR_PERF_SETTINGS_DOMAIN_CPU_EXT,
19     XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT); ③
20 pfnPerfSettingsSetPerformanceLevelEXT(session, XR_PERF_SETTINGS_DOMAIN_GPU_EXT,
21     XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT);
```

① we assume that `instance` and `session` are initialized and their handles are available

② setting performance level to `XR_PERF_SETTINGS_LEVEL_BOOST_EXT` on both CPU and GPU domains



③ going back to the sustainable `XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT`

### Example of using the sustained low level hint for the CPU domain

#### *power reduction sample code*

```
1 extern XrInstance instance; ①
2 extern XrSession session;
3
4 // Get function pointer for xrPerfSettingsSetPerformanceLevelEXT
5 PFN_xrPerfSettingsSetPerformanceLevelEXT pfnPerfSettingsSetPerformanceLevelEXT;
6 CHK_XR(xrGetInstanceProcAddr(instance, "xrPerfSettingsSetPerformanceLevelEXT",
7     (PFN_xrVoidFunction*)(
8     &pfnPerfSettingsSetPerformanceLevelEXT)));
9
10 // before entering a low CPU complexity section
11 pfnPerfSettingsSetPerformanceLevelEXT(session, XR_PERF_SETTINGS_DOMAIN_CPU_EXT,
12     XR_PERF_SETTINGS_LEVEL_SUSTAINED_LOW_EXT);
13
14 // entering the low complexity section
15 // ... running
16 // end of the low complexity section
17
18 pfnPerfSettingsSetPerformanceLevelEXT(session, XR_PERF_SETTINGS_DOMAIN_CPU_EXT,
19     XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT); ③
```

① we assume that `instance` and `session` are initialized and their handles are available

② the developer may choose to only reduce CPU domain and keep the GPU domain at `XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT`

③ going back to the sustainable `XR_PERF_SETTINGS_LEVEL_SUSTAINED_HIGH_EXT` for CPU

### 12.34.3. Receiving Performance Related Notifications

The XR runtime shall provide performance related notifications to the application in the following situations:

- the compositing performance within the runtime has reached a new level, either improved or degraded from the previous one (`subDomain` is set to `XR_PERF_SETTINGS_SUB_DOMAIN_COMPOSITING_EXT`)
- the application rendering performance has reached a new level, either improved or degraded from the previous one (`subDomain` is set to `XR_PERF_SETTINGS_SUB_DOMAIN_RENDERING_EXT`)
- the temperature of the device has reached a new level, either improved or degraded from the previous one (`subDomain` is set to `XR_PERF_SETTINGS_SUB_DOMAIN_THERMAL_EXT`).

When degradation is observed, the application **should** take measures reducing its workload, helping

the compositing or rendering `subDomain` to meet their deadlines, or the thermal `subDomain` to avoid or stop throttling. When improvement is observed, the application can potentially rollback some of its mitigations.

```
// Provided by XR_EXT_performance_settings
typedef struct XrEventDataPerfSettingsEXT {
    XrStructureType          type;
    const void*              next;
    XrPerfSettingsDomainEXT  domain;
    XrPerfSettingsSubDomainEXT subDomain;
    XrPerfSettingsNotificationLevelEXT fromLevel;
    XrPerfSettingsNotificationLevelEXT toLevel;
} XrEventDataPerfSettingsEXT;
```

```
// Provided by XR_EXT_performance_settings
typedef enum XrPerfSettingsSubDomainEXT {
    XR_PERF_SETTINGS_SUB_DOMAIN_COMPOSITING_EXT = 1,
    XR_PERF_SETTINGS_SUB_DOMAIN_RENDERING_EXT = 2,
    XR_PERF_SETTINGS_SUB_DOMAIN_THERMAL_EXT = 3,
    XR_PERF_SETTINGS_SUB_DOMAIN_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsSubDomainEXT;
```

## Compositing Sub-Domain

One of the major functions the runtime shall provide is the timely compositing of the submitted layers in the background. The runtime has to share the CPU and GPU system resources for this operation with the application. Since this is extremely time sensitive - the head room is only a few milliseconds - the runtime may have to ask the application via notifications to cooperate and relinquish some usage of the indicated resource (CPU or GPU domain). Performance issues in this area that the runtime notices are notified to the application with the `subDomain` set to `XR_PERF_SETTINGS_SUB_DOMAIN_COMPOSITING_EXT`.

## Rendering Sub-Domain

The application submits rendered layers to the runtime for compositing. Performance issues in this area that the runtime notices (i.e. missing submission deadlines) are notified to the application with the `subDomain` set to `XR_PERF_SETTINGS_SUB_DOMAIN_RENDERING_EXT`.

## Thermal Sub-Domain

XR applications run at a high-performance level during long periods of time, across a game or an entire movie session. As form factors shrink, especially on mobile solutions, the risk of reaching die thermal runaway or reaching the limits on skin and battery temperatures increases. When thermal limits are reached, the device mitigates the heat generation leading to severe performance reductions, which greatly affects user experience (dropped frames, high latency).

Better than dropping frames when it is too late, pro-active measures from the application should be encouraged.

The performance notification with the `subDomain` set to `XR_PERF_SETTINGS_SUB_DOMAIN_THERMAL_EXT` provides an early warning allowing the application to take mitigation actions.

### Notification level definition

The levels are defined as follows:

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsNotificationLevelEXT {
    XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT = 0,
    XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT = 25,
    XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT = 75,
    XR_PERF_SETTINGS_NOTIFICATION_LEVEL_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsNotificationLevelEXT;
```

- `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` notifies that the sub-domain has reached a level where no further actions other than currently applied are necessary.
- `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` notifies that the sub-domain has reached an early warning level where the application should start proactive mitigation actions with the goal to return to the `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` level.
- `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` notifies that the sub-domain has reached a critical level with significant performance degradation. The application should take drastic mitigation action.

The above definitions summarize the broad interpretation of the notification levels, however sub-domain specific definitions of each level and their transitions are specified below:

- `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT`
  - For the compositing sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` indicates that the composition headroom is consistently being met with sufficient margin.  
Getting into `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` from `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` indicates that the composition headroom was consistently **met with sufficient margin during a sufficient time period**.
  - For the rendering sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` indicates that frames are being submitted in time to be used by the compositor.  
Getting into `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` from `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` indicates that **during a sufficient time period, none** of the due layers was **too late** to be picked up by the compositor.
  - For the thermal sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` indicates that the current load should be sustainable in the near future.  
Getting into `XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT` from

`XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` indicates that the runtime does not presuppose any further temperature mitigation action on the application side, other than the current ones.

- `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT`

- For the compositing sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` indicates that the compositing headroom of the current frame was met but the margin is considered insufficient by the runtime, and the application **should** reduce its workload in the notified domain to solve this problem.

Getting `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` into `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` from `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that the compositing deadline was **not missed during a sufficient time period**.

- For the rendering sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` indicates that at least one layer is regularly late to be picked up by the compositor, resulting in a degraded user experience, and that the application should take action to consistently provide frames in a more timely manner.

Getting `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` into `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` from `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that **the runtime has stopped any of its own independent actions** which are tied to the `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` level.

- For the thermal sub-domain, the `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` indicates that the runtime expects the device to overheat under the current load, and that the application should take mitigating action in order to prevent thermal throttling.

Getting `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` into `XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT` from `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that **the underlying system thermal throttling has stopped**.

- `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT`

- For the compositing sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that composition can no longer be maintained under the current workload. The runtime may take independent action that will interfere with the application (e.g. limiting the framerate, ignoring submitted layers, or shutting down the application) in order to correct this problem.

- For the rendering sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that at least one layer is **too often** late to be picked up by the compositor, and consequently the runtime may take independent action that will interfere with the application (e.g. informing the user that the application is not responding, displaying a tracking environment in order to maintain user orientation).

- For the thermal sub-domain, `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that the **underlying system is taking measures, such as thermal throttling** to reduce the temperature, impacting the XR experience.

Leaving `XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT` indicates that any mitigating actions by the runtime (e.g. down-clocking the device to stay within thermal limits) have ended.

### xrPerfSettingsSetPerformanceLevelEXT

```
// Provided by XR_EXT_performance_settings
XrResult xrPerfSettingsSetPerformanceLevelEXT(
    XrSession session,
    XrPerfSettingsDomainEXT domain,
    XrPerfSettingsLevelEXT level);
```

#### Parameter Descriptions

- **session** is a valid [XrSession](#) handle.
- **domain**: the processing domain for which the level hint is applied
- **level**: the level hint to be applied

#### Valid Usage (Implicit)

- The [XR\\_EXT\\_performance\\_settings](#) extension **must** be enabled prior to calling [xrPerfSettingsSetPerformanceLevelEXT](#)
- **session** **must** be a valid [XrSession](#) handle
- **domain** **must** be a valid [XrPerfSettingsDomainEXT](#) value
- **level** **must** be a valid [XrPerfSettingsLevelEXT](#) value

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST

Refer to [Performance level hint definition](#) for the definition of the level enumerations.

## XrEventDataPerformanceSettingsEXT

```
// Provided by XR_EXT_performance_settings
typedef struct XrEventDataPerfSettingsEXT {
    XrStructureType          type;
    const void*              next;
    XrPerfSettingsDomainEXT  domain;
    XrPerfSettingsSubDomainEXT subDomain;
    XrPerfSettingsNotificationLevelEXT fromLevel;
    XrPerfSettingsNotificationLevelEXT toLevel;
} XrEventDataPerfSettingsEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `domain` : processing domain in which a threshold has been crossed
- `subDomain` : system area in which a threshold has been crossed
- `fromLevel` : enumerated notification level which has been exited
- `toLevel` : enumerated notification level which has been entered

## Valid Usage (Implicit)

- The [XR\\_EXT\\_performance\\_settings](#) extension **must** be enabled prior to using [XrEventDataPerfSettingsEXT](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_PERF_SETTINGS_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsDomainEXT {
    XR_PERF_SETTINGS_DOMAIN_CPU_EXT = 1,
    XR_PERF_SETTINGS_DOMAIN_GPU_EXT = 2,
    XR_PERF_SETTINGS_DOMAIN_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsDomainEXT;
```

```
// Provided by XR_EXT_performance_settings
typedef enum XrPerfSettingsSubDomainEXT {
    XR_PERF_SETTINGS_SUB_DOMAIN_COMPOSITING_EXT = 1,
    XR_PERF_SETTINGS_SUB_DOMAIN_RENDERING_EXT = 2,
    XR_PERF_SETTINGS_SUB_DOMAIN_THERMAL_EXT = 3,
    XR_PERF_SETTINGS_SUB_DOMAIN_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsSubDomainEXT;
```

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsNotificationLevelEXT {
    XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT = 0,
    XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT = 25,
    XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT = 75,
    XR_PERF_SETTINGS_NOTIFICATION_LEVEL_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsNotificationLevelEXT;
```

## Version History

- Revision 1, 2017-11-30 (Armelle Laine)
- Revision 2, 2021-04-13 (Rylie Pavlik)
  - Correctly show function pointer retrieval in sample code
  - Fix sample code callouts
- Revision 3, 2021-04-14 (Rylie Pavlik)
  - Fix missing error code
- Revision 4, 2022-10-26 (Rylie Pavlik)
  - Update XML markup to correct the generated valid usage

## 12.35. XR\_EXT\_plane\_detection

### Name String

`XR_EXT_plane_detection`

### Extension Type

Instance extension

### Registered Extension Number

430

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-06-26

### Contributors

Aitor Font, Qualcomm



Daniel Guttenberg, Qualcomm  
Maximilian Mayer, Qualcomm  
Martin Renschler, Qualcomm  
Karthik Nagarajan, Qualcomm  
Ron Bessems, Magic Leap  
Karthik Kadappan, Magic Leap

### 12.35.1. Overview

This extension enables applications to detect planes in the scene.

### 12.35.2. Runtime support

To determine if this runtime supports detecting planes `xrGetSystemProperties` can be used.

`XrSystemPlaneDetectionPropertiesEXT` provides information on the features supported by the runtime.

```
// Provided by XR_EXT_plane_detection
typedef struct XrSystemPlaneDetectionPropertiesEXT {
    XrStructureType          type;
    void*                    next;
    XrPlaneDetectionCapabilityFlagsEXT supportedFeatures;
} XrSystemPlaneDetectionPropertiesEXT;
```

#### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `supportedFeatures` is a bitfield, with bit masks defined in `XrPlaneDetectionCapabilityFlagBitsEXT`.

#### Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to using `XrSystemPlaneDetectionPropertiesEXT`
- `type` **must** be `XR_TYPE_SYSTEM_PLANE_DETECTION_PROPERTIES_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrSystemPlaneDetectionPropertiesEXT::supportedFeatures` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in `XrPlaneDetectionCapabilityFlagBitsEXT`.

```
// Provided by XR_EXT_plane_detection
typedef XrFlags64 XrPlaneDetectionCapabilityFlagsEXT;
```

Valid bits for `XrPlaneDetectionCapabilityFlagsEXT` are defined by `XrPlaneDetectionCapabilityFlagBitsEXT`, which is specified as:

```
// Flag bits for XrPlaneDetectionCapabilityFlagsEXT
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_PLANE_DETECTION_BIT_EXT = 0x00000001;
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_PLANE_HOLES_BIT_EXT = 0x00000002;
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_CEILING_BIT_EXT = 0x00000004;
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_FLOOR_BIT_EXT = 0x00000008;
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_WALL_BIT_EXT = 0x00000010;
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_PLATFORM_BIT_EXT = 0x00000020;
static const XrPlaneDetectionCapabilityFlagsEXT
XR_PLANE_DETECTION_CAPABILITY_ORIENTATION_BIT_EXT = 0x00000040;
```

The flag bits have the following meanings:

## Flag Descriptions

- `XR_PLANE_DETECTION_CAPABILITY_PLANE_DETECTION_BIT_EXT` — plane detection is supported
- `XR_PLANE_DETECTION_CAPABILITY_PLANE_HOLES_BIT_EXT` — polygon buffers for holes in planes can be generated
- `XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_CEILING_BIT_EXT` — plane detection supports ceiling semantic classification
- `XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_FLOOR_BIT_EXT` — plane detection supports floor semantic classification
- `XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_WALL_BIT_EXT` — plane detection supports wall semantic classification
- `XR_PLANE_DETECTION_CAPABILITY_SEMANTIC_PLATFORM_BIT_EXT` — plane detection supports platform semantic classification (for example table tops)
- `XR_PLANE_DETECTION_CAPABILITY_ORIENTATION_BIT_EXT` — plane detection supports plane orientation classification. If not supported planes are always classified as ARBITRARY.

### 12.35.3. Create a plane detection handle

```
// Provided by XR_EXT_plane_detection
XR_DEFINE_HANDLE(XrPlaneDetectorEXT)
```

The `XrPlaneDetectorEXT` handle represents the resources for detecting one or more planes.

An application **may** create separate `XrPlaneDetectorEXT` handles for different sets of planes. This handle **can** be used to detect planes using other functions in this extension.

Plane detection provides locations of planes in the scene.

The `xrCreatePlaneDetectorEXT` function is defined as:

```
// Provided by XR_EXT_plane_detection
XrResult xrCreatePlaneDetectorEXT(
    XrSession session,
    const XrPlaneDetectorCreateInfoEXT* createInfo,
    XrPlaneDetectorEXT* planeDetector);
```

## Parameter Descriptions

- `session` is an [XrSession](#) in which the plane detection will be active.
- `createInfo` is the [XrPlaneDetectorCreateInfoEXT](#) used to specify the plane detection.
- `planeDetector` is the returned [XrPlaneDetectorEXT](#) handle.

An application creates an [XrPlaneDetectorEXT](#) handle using [xrCreatePlaneDetectorEXT](#) function.

If the system does not support plane detection, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreatePlaneDetectorEXT](#).

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to calling [xrCreatePlaneDetectorEXT](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrPlaneDetectorCreateInfoEXT](#) structure
- `planeDetector` **must** be a pointer to an [XrPlaneDetectorEXT](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_PLANE_DETECTION_PERMISSION_DENIED_EXT`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrPlaneDetectorCreateInfoEXT` structure is defined as:

```
// Provided by XR_EXT_plane_detection
typedef struct XrPlaneDetectorCreateInfoEXT {
    XrStructureType      type;
    const void*         next;
    XrPlaneDetectorFlagsEXT flags;
} XrPlaneDetectorCreateInfoEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` **must** be a valid combination of `XrPlaneDetectorFlagsEXT` flags or zero.

The `XrPlaneDetectorCreateInfoEXT` structure describes the information to create an

[XrPlaneDetectorEXT](#) handle.

### Valid Usage (Implicit)

- The [XR\\_EXT\\_plane\\_detection](#) extension **must** be enabled prior to using [XrPlaneDetectorCreateInfoEXT](#)
- **type** **must** be `XR_TYPE_PLANE_DETECTOR_CREATE_INFO_EXT`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **flags** **must** be `0` or a valid combination of [XrPlaneDetectorFlagBitsEXT](#) values

The [XrPlaneDetectorCreateInfoEXT::flags](#) member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in [XrPlaneDetectorFlagBitsEXT](#).

```
// Provided by XR_EXT_plane_detection
typedef XrFlags64 XrPlaneDetectorFlagsEXT;
```

Valid bits for [XrPlaneDetectorFlagsEXT](#) are defined by [XrPlaneDetectorFlagBitsEXT](#), which is specified as:

```
// Flag bits for XrPlaneDetectorFlagsEXT
static const XrPlaneDetectorFlagsEXT XR_PLANE_DETECTOR_ENABLE_CONTOUR_BIT_EXT =
0x00000001;
```

The flag bits have the following meanings:

### Flag Descriptions

- `XR_PLANE_DETECTOR_ENABLE_CONTOUR_BIT_EXT` — populate the plane contour information

The [xrDestroyPlaneDetectorEXT](#) function is defined as:

```
// Provided by XR_EXT_plane_detection
XrResult xrDestroyPlaneDetectorEXT(
    XrPlaneDetectorEXT                planeDetector);
```

## Parameter Descriptions

- `planeDetector` is an `XrPlaneDetectorEXT` previously created by `xrCreatePlaneDetectorEXT`.

`xrDestroyPlaneDetectorEXT` function releases the `planeDetector` and the underlying resources when finished with plane detection experiences.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to calling `xrDestroyPlaneDetectorEXT`
- `planeDetector` **must** be a valid `XrPlaneDetectorEXT` handle

## Thread Safety

- Access to `planeDetector`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`

## 12.35.4. Detecting planes

The `xrBeginPlaneDetectionEXT` function is defined as:

```
// Provided by XR_EXT_plane_detection
XrResult xrBeginPlaneDetectionEXT(
    XrPlaneDetectorEXT                planeDetector,
    const XrPlaneDetectorBeginInfoEXT* beginInfo);
```

## Parameter Descriptions

- `planeDetector` is an `XrPlaneDetectorEXT` previously created by `xrCreatePlaneDetectorEXT`.
- `beginInfo` is a pointer to `XrPlaneDetectorBeginInfoEXT` containing plane detection parameters.

The `xrBeginPlaneDetectionEXT` function begins the detection of planes in the scene. Detecting planes in a scene is an asynchronous operation. `xrGetPlaneDetectionStateEXT` **can** be used to determine if the query has finished. Once it has finished the results **may** be retrieved via `xrGetPlaneDetectionsEXT`. If a detection has already been started on a plane detector handle, calling `xrBeginPlaneDetectionEXT` again on the same handle will cancel the operation in progress and start a new detection with the new filter parameters.

The bounding volume is resolved and fixed relative to LOCAL space at the time of the call to `xrBeginPlaneDetectionEXT` using `XrPlaneDetectorBeginInfoEXT::baseSpace`, `XrPlaneDetectorBeginInfoEXT::time`, `XrPlaneDetectorBeginInfoEXT::boundingBoxPose` and `XrPlaneDetectorBeginInfoEXT::boundingBoxExtent`. The runtime **must** resolve the location defined by `XrPlaneDetectorBeginInfoEXT::baseSpace` at the time of the call. The `XrPlaneDetectorBeginInfoEXT::boundingBoxPose` is the pose of the center of the box defined by `XrPlaneDetectorBeginInfoEXT::boundingBoxExtent`.

The runtime **must** return `XR_ERROR_SPACE_NOT_LOCATABLE_EXT` if the `XrPlaneDetectorBeginInfoEXT::baseSpace` is not locatable at the time of the call.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to calling `xrBeginPlaneDetectionEXT`
- `planeDetector` **must** be a valid `XrPlaneDetectorEXT` handle
- `beginInfo` **must** be a pointer to a valid `XrPlaneDetectorBeginInfoEXT` structure



## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_TIME\_INVALID
- XR\_ERROR\_SPACE\_NOT\_LOCATABLE\_EXT
- XR\_ERROR\_POSE\_INVALID

The `XrPlaneDetectorBeginInfoEXT` structure describes the information to detect planes.

```
// Provided by XR_EXT_plane_detection
typedef struct XrPlaneDetectorBeginInfoEXT {
    XrStructureType                type;
    const void*                    next;
    XrSpace                        baseSpace;
    XrTime                         time;
    uint32_t                       orientationCount;
    const XrPlaneDetectorOrientationEXT* orientations;
    uint32_t                       semanticTypeCount;
    const XrPlaneDetectorSemanticTypeEXT* semanticTypes;
    uint32_t                       maxPlanes;
    float                          minArea;
    XrPosef                        boundingBoxPose;
    XrExtent3DfEXT                 boundingBoxExtent;
} XrPlaneDetectorBeginInfoEXT;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **baseSpace** is the [XrSpace](#) that the **boundingBoxPose** is defined in.
- **time** is an [XrTime](#) at which to detect the planes.
- **orientationCount** the number of elements in the **orientations**.
- **orientations** an array of [XrPlaneDetectorOrientationEXT](#). If this field is null no orientation filtering is applied. If any orientations are present only planes with any of the orientation listed are returned.
- **semanticTypeCount** the number of elements in the **semanticTypes**.
- **semanticTypes** an array of [XrPlaneDetectorSemanticTypeEXT](#). If this field is null no semantic type filtering is applied. If any semantic types are present only planes with matching semantic types are returned.
- **maxPlanes** is the maximum number of planes the runtime **may** return. This number **must** be larger than 0. If the number is 0 the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.
- **minArea** is the minimum area in square meters a plane **must** have to be returned. A runtime **may** have a lower limit under which planes are not detected regardless of **minArea** and silently drop planes lower than the internal minimum.
- **boundingBoxPose** is the pose of the center of the bounding box of the volume to use for detection in **baseSpace**.
- **boundingBoxExtent** is the extent of the bounding box to use for detection. If any part of a plane falls within the bounding box it **should** be considered for inclusion subject to the other filters. This means that planes **may** extend beyond the bounding box. A runtime **may** have an upper limit on the detection range and silently clip the results to that internally.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to using `XrPlaneDetectorBeginInfoEXT`
- `type` **must** be `XR_TYPE_PLANE_DETECTOR_BEGIN_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `baseSpace` **must** be a valid `XrSpace` handle
- If `orientationCount` is not `0`, `orientations` **must** be a pointer to an array of `orientationCount` valid `XrPlaneDetectorOrientationEXT` values
- If `semanticTypeCount` is not `0`, `semanticTypes` **must** be a pointer to an array of `semanticTypeCount` valid `XrPlaneDetectorSemanticTypeEXT` values

The `xrGetPlaneDetectionStateEXT` function is defined as:

```
// Provided by XR_EXT_plane_detection
XrResult xrGetPlaneDetectionStateEXT(
    XrPlaneDetectorEXT          planeDetector,
    XrPlaneDetectionStateEXT*  state);
```

## Parameter Descriptions

- `planeDetector` is an `XrPlaneDetectorEXT` previously created by `xrCreatePlaneDetectorEXT`.
- `state` is a pointer to `XrPlaneDetectionStateEXT`.

The `xrGetPlaneDetectionStateEXT` function retrieves the state of the plane query and **must** be called before calling `xrGetPlaneDetectionsEXT`.

If the plane detection has not yet finished `state` **must** be `XR_PLANE_DETECTION_STATE_PENDING_EXT`. If the plane detection has finished `state` **must** be `XR_PLANE_DETECTION_STATE_DONE_EXT`. If no plane detection was previously started `XR_PLANE_DETECTION_STATE_NONE_EXT` **must** be returned. For all three states the function **must** return `XR_SUCCESS`.

When a query error occurs the function **must** return `XR_SUCCESS` and the appropriate error state value **must** be set.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to calling `xrGetPlaneDetectionStateEXT`
- `planeDetector` **must** be a valid `XrPlaneDetectorEXT` handle
- `state` **must** be a pointer to an `XrPlaneDetectionStateEXT` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrGetPlaneDetectionsEXT` function is defined as:

```
// Provided by XR_EXT_plane_detection
XrResult xrGetPlaneDetectionsEXT(
    XrPlaneDetectorEXT                planeDetector,
    const XrPlaneDetectorGetInfoEXT*  info,
    XrPlaneDetectorLocationsEXT*     locations);
```

## Parameter Descriptions

- `planeDetector` is an `XrPlaneDetectorEXT` previously created by `xrCreatePlaneDetectorEXT`.
- `info` is a pointer to `XrPlaneDetectorGetInfoEXT`.
- `locations` is a pointer to `XrPlaneDetectorLocationsEXT` receiving the returned plane locations.

`xrGetPlaneDetectionsEXT` **must** return `XR_ERROR_CALL_ORDER_INVALID` if the detector state reported by `xrGetPlaneDetectionStateEXT` is not `XR_PLANE_DETECTION_STATE_DONE_EXT` for the current query started by `xrBeginPlaneDetectionEXT`.

If the `XrPlaneDetectorGetInfoEXT::baseSpace` is not locatable `XR_ERROR_SPACE_NOT_LOCATABLE_EXT` **must** be returned.

Once `xrBeginPlaneDetectionEXT` is called again, the previous results for that handle are no longer available. The application **should** cache them before calling `xrBeginPlaneDetectionEXT` again if it needs access to that data while waiting for updated detection results.

Upon the completion of a detection cycle (`xrBeginPlaneDetectionEXT`, `xrGetPlaneDetectionStateEXT` to `xrGetPlaneDetectionsEXT`) the runtime **must** keep a snapshot of the plane data and no data **may** be modified. Calling `xrGetPlaneDetectionsEXT` multiple times with the same `baseSpace` and time **must** return the same plane pose data.

The current snapshot, if any, **must** be discarded upon calling `xrBeginPlaneDetectionEXT`.

If the `XrEventDataReferenceSpaceChangePending` is queued and the `changeTime` elapsed while the application is holding cached data the application **may** use the event data to adjusted poses accordingly.

### Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to calling `xrGetPlaneDetectionsEXT`
- `planeDetector` **must** be a valid `XrPlaneDetectorEXT` handle
- `info` **must** be a pointer to a valid `XrPlaneDetectorGetInfoEXT` structure
- `locations` **must** be a pointer to an `XrPlaneDetectorLocationsEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`
- `XR_ERROR_SPACE_NOT_LOCATABLE_EXT`
- `XR_ERROR_CALL_ORDER_INVALID`

`XrPlaneDetectorGetInfoEXT` structure contains the information required to retrieve the detected planes.

```
// Provided by XR_EXT_plane_detection
typedef struct XrPlaneDetectorGetInfoEXT {
    XrStructureType    type;
    const void*        next;
    XrSpace             baseSpace;
    XrTime             time;
} XrPlaneDetectorGetInfoEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `baseSpace` the plane pose will be relative to this `XrSpace` at `time`.
- `time` is the `XrTime` at which to evaluate the coordinates relative to the `baseSpace`.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to using `XrPlaneDetectorGetInfoEXT`
- `type` **must** be `XR_TYPE_PLANE_DETECTOR_GET_INFO_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `baseSpace` **must** be a valid `XrSpace` handle

`XrPlaneDetectorLocationsEXT` structure contains information on the detected planes.

```
// Provided by XR_EXT_plane_detection
typedef struct XrPlaneDetectorLocationsEXT {
    XrStructureType      type;
    void*                next;
    uint32_t             planeLocationCapacityInput;
    uint32_t             planeLocationCountOutput;
    XrPlaneDetectorLocationEXT* planeLocations;
} XrPlaneDetectorLocationsEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `planeLocationCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `planeLocationCountOutput` is the number of planes, or the required capacity in the case that `planeLocationCapacityInput` is insufficient.
- `planeLocations` is an array of `XrPlaneDetectorLocationEXT`. It **can** be `NULL` if `planeLocationCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `planeLocations` size.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to using `XrPlaneDetectorLocationsEXT`
- `type` **must** be `XR_TYPE_PLANE_DETECTOR_LOCATIONS_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `planeLocationCapacityInput` is not `0`, `planeLocations` **must** be a pointer to an array of `planeLocationCapacityInput` `XrPlaneDetectorLocationEXT` structures

`XrPlaneDetectorLocationEXT` structure describes the position and orientation of a plane.

```
// Provided by XR_EXT_plane_detection
typedef struct XrPlaneDetectorLocationEXT {
    XrStructureType          type;
    void*                    next;
    uint64_t                 planeId;
    XrSpaceLocationFlags     locationFlags;
    XrPosef                  pose;
    XrExtent2Df              extents;
    XrPlaneDetectorOrientationEXT orientation;
    XrPlaneDetectorSemanticTypeEXT semanticType;
    uint32_t                 polygonBufferCount;
} XrPlaneDetectorLocationEXT;
```



## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `planeId` is a `uint64_t` unique identifier of the plane. The `planeId` **should** remain the same for the duration of the `XrPlaneDetectorEXT` handle for a physical plane. A runtime on occasion **may** assign a different id to the same physical plane, for example when several planes merge into one plane. `planeId` **must** remain valid until the next call to `xrBeginPlaneDetectionEXT` or `xrDestroyPlaneDetectorEXT`. This id is used by `xrGetPlanePolygonBufferEXT`.
- `locationFlags` is a bitfield, with bit masks defined in `XrSpaceLocationFlagBits`, to indicate which members contain valid data. If none of the bits are set, no other fields in this structure **should** be considered to be valid or meaningful.
- `pose` is an `XrPosef` defining the position and orientation of the origin of a plane within the reference frame of the corresponding `XrPlaneDetectorGetInfoEXT::baseSpace`.
- `extents` is the extent of the plane along the x-axis (width) and z-axis (height) centered on the `pose`.
- `orientation` is the detected orientation of the plane.
- `semanticType` `XrPlaneDetectorSemanticTypeEXT` type of the plane.
- `polygonBufferCount` is the number of polygon buffers associated with this plane. If this is zero no polygon buffer was generated. The first polygon buffer is always the outside contour. If contours are requested with `XR_PLANE_DETECTOR_ENABLE_CONTOUR_BIT_EXT` this value **must** always be at least 1.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to using `XrPlaneDetectorLocationEXT`
- `type` **must** be `XR_TYPE_PLANE_DETECTOR_LOCATION_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `locationFlags` **must** be `0` or a valid combination of `XrSpaceLocationFlagBits` values
- If `orientation` is not `0`, `orientation` **must** be a valid `XrPlaneDetectorOrientationEXT` value
- If `semanticType` is not `0`, `semanticType` **must** be a valid `XrPlaneDetectorSemanticTypeEXT` value

The `XrPlaneDetectorOrientationEXT` enumeration identifies the different general categories of orientations of detected planes.

```
// Provided by XR_EXT_plane_detection
typedef enum XrPlaneDetectorOrientationEXT {
    XR_PLANE_DETECTOR_ORIENTATION_HORIZONTAL_UPWARD_EXT = 0,
    XR_PLANE_DETECTOR_ORIENTATION_HORIZONTAL_DOWNWARD_EXT = 1,
    XR_PLANE_DETECTOR_ORIENTATION_VERTICAL_EXT = 2,
    XR_PLANE_DETECTOR_ORIENTATION_ARBITRARY_EXT = 3,
    XR_PLANE_DETECTOR_ORIENTATION_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPlaneDetectorOrientationEXT;
```

The enums have the following meanings:

| Enum   | Description   |
|--|---|
| <code>XR_PLANE_DETECTOR_ORIENTATION_HORIZONTAL_UPWARD_EXT</code>   | The detected plane is horizontal and faces upward (e.g. floor).                   |
| <code>XR_PLANE_DETECTOR_ORIENTATION_HORIZONTAL_DOWNWARD_EXT</code> | The detected plane is horizontal and faces downward (e.g. ceiling).               |
| <code>XR_PLANE_DETECTOR_ORIENTATION_VERTICAL_EXT</code>            | The detected plane is vertical (e.g. wall).                                       |
| <code>XR_PLANE_DETECTOR_ORIENTATION_ARBITRARY_EXT</code>           | The detected plane has an arbitrary, non-vertical and non-horizontal orientation. |

The `XrPlaneDetectorSemanticTypeEXT` enumeration identifies the different semantic types of detected planes.

```
// Provided by XR_EXT_plane_detection
typedef enum XrPlaneDetectorSemanticTypeEXT {
    XR_PLANE_DETECTOR_SEMANTIC_TYPE_UNDEFINED_EXT = 0,
    XR_PLANE_DETECTOR_SEMANTIC_TYPE_CEILING_EXT = 1,
    XR_PLANE_DETECTOR_SEMANTIC_TYPE_FLOOR_EXT = 2,
    XR_PLANE_DETECTOR_SEMANTIC_TYPE_WALL_EXT = 3,
    XR_PLANE_DETECTOR_SEMANTIC_TYPE_PLATFORM_EXT = 4,
    XR_PLANE_DETECTOR_SEMANTIC_TYPE_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPlaneDetectorSemanticTypeEXT;
```

The enums have the following meanings:

| Enum   | Description                                    |
|--|--|
| <code>XR_PLANE_DETECTOR_SEMANTIC_TYPE_UNDEFINED_EXT</code> | The runtime was unable to classify this plane. |
| <code>XR_PLANE_DETECTOR_SEMANTIC_TYPE_CEILING_EXT</code>   | The detected plane is a ceiling.               |
| <code>XR_PLANE_DETECTOR_SEMANTIC_TYPE_FLOOR_EXT</code>     | The detected plane is a floor.                 |

| Enum  | Description                                     |
|---|---|
| <code>XR_PLANE_DETECTOR_SEMANTIC_TYPE_WALL_EXT</code>     | The detected plane is a wall.                   |
| <code>XR_PLANE_DETECTOR_SEMANTIC_TYPE_PLATFORM_EXT</code> | The detected plane is a platform, like a table. |

The `XrPlaneDetectionStateEXT` enumeration identifies the possible states of the plane detector.

```
// Provided by XR_EXT_plane_detection
typedef enum XrPlaneDetectionStateEXT {
    XR_PLANE_DETECTION_STATE_NONE_EXT = 0,
    XR_PLANE_DETECTION_STATE_PENDING_EXT = 1,
    XR_PLANE_DETECTION_STATE_DONE_EXT = 2,
    XR_PLANE_DETECTION_STATE_ERROR_EXT = 3,
    XR_PLANE_DETECTION_STATE_FATAL_EXT = 4,
    XR_PLANE_DETECTION_STATE_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPlaneDetectionStateEXT;
```

### Enumerant Descriptions

- `XR_PLANE_DETECTION_STATE_NONE_EXT` - The plane detector is not actively looking for planes; call `xrBeginPlaneDetectionEXT` to start detection.
- `XR_PLANE_DETECTION_STATE_PENDING_EXT` - This plane detector is currently looking for planes but not yet ready with results; call `xrGetPlaneDetectionsEXT` again, or call `xrBeginPlaneDetectionEXT` to restart with new filter parameters.
- `XR_PLANE_DETECTION_STATE_DONE_EXT` - This plane detector has finished and results **may** now be retrieved. The results are valid until `xrBeginPlaneDetectionEXT` or `xrDestroyPlaneDetectorEXT` are called.
- `XR_PLANE_DETECTION_STATE_ERROR_EXT` - An error occurred. The query **may** be tried again.
- `XR_PLANE_DETECTION_STATE_FATAL_EXT` - An error occurred. The query **must** not be tried again.

### 12.35.5. Read plane polygon vertices

The `xrGetPlanePolygonBufferEXT` function is defined as:

```
// Provided by XR_EXT_plane_detection
XrResult xrGetPlanePolygonBufferEXT(
    XrPlaneDetectorEXT          planeDetector,
    uint64_t                    planeId,
    uint32_t                    polygonBufferIndex,
    XrPlaneDetectorPolygonBufferEXT* polygonBuffer);
```

## Parameter Descriptions

- `planeDetector` is an `XrPlaneDetectorEXT` previously created by `xrCreatePlaneDetectorEXT`.
- `planeId` is the `XrPlaneDetectorLocationEXT::planeId`.
- `polygonBufferIndex` is the index of the polygon contour buffer to retrieve. This **must** be a number from 0 to `XrPlaneDetectorLocationEXT::polygonBufferCount - 1`. Index 0 retrieves the outside contour, larger indexes retrieve holes in the plane.
- `polygonBuffer` is a pointer to `XrPlaneDetectorPolygonBufferEXT` receiving the returned plane polygon buffer.

The `xrGetPlanePolygonBufferEXT` function retrieves the plane's polygon buffer for the given `planeId` and `polygonBufferIndex`. Calling `xrGetPlanePolygonBufferEXT` with `polygonBufferIndex` equal to 0 **must** return the outside contour, if available. Calls with non-zero indices less than `XrPlaneDetectorLocationEXT::polygonBufferCount` **must** return polygons corresponding to holes in the plane. This feature **may** not be supported by all runtimes, check the `XrSystemPlaneDetectionPropertiesEXT::supportedFeatures` for support.

Outside contour polygon vertices **must** be ordered in counter clockwise order. Vertices of holes **must** be ordered in clockwise order. The right-hand rule is used to determine the direction of the normal of this plane. The polygon contour data is relative to the pose of the plane and coplanar with it.

This function only retrieves polygons, which means that it needs to be converted to a regular mesh to be rendered.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to calling `xrGetPlanePolygonBufferEXT`
- `planeDetector` **must** be a valid `XrPlaneDetectorEXT` handle
- `polygonBuffer` **must** be a pointer to an `XrPlaneDetectorPolygonBufferEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

`XrPlaneDetectorPolygonBufferEXT` is an input/output structure for reading plane contour polygon vertices.

```
// Provided by XR_EXT_plane_detection
typedef struct XrPlaneDetectorPolygonBufferEXT {
    XrStructureType    type;
    void*              next;
    uint32_t           vertexCapacityInput;
    uint32_t           vertexCountOutput;
    XrVector2f*        vertices;
} XrPlaneDetectorPolygonBufferEXT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `vertexCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `vertexCountOutput` is the count of `vertices` written, or the required capacity in the case that `vertexCapacityInput` is insufficient.
- `vertices` is an array of `XrVector2f` that **must** be filled by the runtime with the positions of the polygon vertices relative to the plane's pose.

## Valid Usage (Implicit)

- The `XR_EXT_plane_detection` extension **must** be enabled prior to using `XrPlaneDetectorPolygonBufferEXT`
- `type` **must** be `XR_TYPE_PLANE_DETECTOR_POLYGON_BUFFER_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `vertexCapacityInput` is not `0`, `vertices` **must** be a pointer to an array of `vertexCapacityInput` `XrVector2f` structures

The `XrExtent3DfEXT` structure is defined as:

```
// Provided by XR_EXT_plane_detection
// XrExtent3DfEXT is an alias for XrExtent3Df
typedef struct XrExtent3Df {
    float    width;
    float    height;
    float    depth;
} XrExtent3Df;

typedef XrExtent3Df XrExtent3DfEXT;
```

## Member Descriptions

- `width` the floating-point width of the extent.
- `height` the floating-point height of the extent.
- `depth` the floating-point depth of the extent.

The `XrExtent3DfEXT` structure describes a axis aligned three-dimensional floating-point extent: This structure is used for component values that **may** be fractional (floating-point). If used to represent physical distances, values **must** be in meters.

The `width` (X), `height` (Y) and `depth` (Z) values **must** be non-negative.

### 12.35.6. Example code for locating planes

The following example code demonstrates how to detect planes relative to a local space.

```
XrInstance instance; // previously initialized
```

```

XrSystemId systemId; // previously initialized
XrSession session; // previously initialized
XrSpace localSpace; // previously initialized, e.g. from
                    // XR_REFERENCE_SPACE_TYPE_LOCAL
XrSpace viewSpace; // previously initialized, e.g. from
                   // XR_REFERENCE_SPACE_TYPE_VIEW

// The function pointers are previously initialized using
// xrGetInstanceProcAddr.
PFN_xrCreatePlaneDetectorEXT xrCreatePlaneDetectorEXT; // previously initialized
PFN_xrBeginPlaneDetectionEXT xrBeginPlaneDetectionEXT; // previously initialized
PFN_xrGetPlaneDetectionStateEXT xrGetPlaneDetectionStateEXT; // previously initialized
PFN_xrGetPlaneDetectionsEXT xrGetPlaneDetectionsEXT; // previously initialized
PFN_xrGetPlanePolygonBufferEXT xrGetPlanePolygonBufferEXT; // previously initialized

XrSystemProperties properties{XR_TYPE_SYSTEM_PROPERTIES};
XrSystemPlaneDetectionPropertiesEXT
planeDetectionProperties{XR_TYPE_SYSTEM_PLANE_DETECTION_PROPERTIES_EXT};
properties.next = &planeDetectionProperties;

CHK_XR(xrGetSystemProperties(instance, systemId, &properties));
if (!(planeDetectionProperties.supportedFeatures &
XR_PLANE_DETECTION_CAPABILITY_PLANE_DETECTION_BIT_EXT )) {
    // plane detection is not supported.
    return;
}

// Create a plane detection
XrPlaneDetectorEXT planeDetector{};
{
    XrPlaneDetectorCreateInfoEXT createInfo{ XR_TYPE_PLANE_DETECTOR_CREATE_INFO_EXT };
    createInfo.flags = XR_PLANE_DETECTOR_ENABLE_CONTOUR_BIT_EXT;
    CHK_XR(xrCreatePlaneDetectorEXT(session, &createInfo, &planeDetector));
}

bool queryRunning = false;

std::vector<XrPlaneDetectorOrientationEXT> orientations;
orientations.push_back(XR_PLANE_DETECTOR_ORIENTATION_HORIZONTAL_UPWARD_EXT);
orientations.push_back(XR_PLANE_DETECTOR_ORIENTATION_HORIZONTAL_DOWNWARD_EXT);

std::vector<XrPlaneDetectorLocationEXT> cachedPlaneLocations;

auto processPlanes = [&](const XrTime time) {

    if (!queryRunning) {

```

```

    XrPlaneDetectorBeginInfoEXT beginInfo{ XR_TYPE_PLANE_DETECTOR_BEGIN_INFO_EXT };
    XrPosef pose{};
    XrExtent3DfEXT extents = {10.0f, 10.0f, 10.0f};
    pose.orientation.w = 1.0f;
    beginInfo.baseSpace = viewSpace;
    beginInfo.time = time;
    beginInfo.boundingBoxPose = pose;
    beginInfo.boundingBoxExtent = extents;
    beginInfo.orientationCount = (uint32_t)orientations.size();
    beginInfo.orientations = orientations.data();

    CHK_XR(xrBeginPlaneDetectionEXT(planeDetector, &beginInfo));
    queryRunning = true;
    return;
} else {
    XrPlaneDetectionStateEXT planeDetectionState;
    if (xrGetPlaneDetectionStateEXT(planeDetector, &planeDetectionState)!=XR_SUCCESS)
    {
        queryRunning = false;
        return;
    }

    switch(planeDetectionState) {
    case XR_PLANE_DETECTION_STATE_DONE_EXT:
        // query has finished, process the results.
        break;
    case XR_PLANE_DETECTION_STATE_ERROR_EXT:
        // something temporary went wrong, just
        // retry
        queryRunning = false;
        return;
    case XR_PLANE_DETECTION_STATE_FATAL_EXT:
        // there was something wrong with the query
        // do not retry.
        // exit();
        return;
    case XR_PLANE_DETECTION_STATE_PENDING_EXT:
        // query is still processing, come back on the next loop.
        return;
    default:
        // restart the query.
        queryRunning = false;
        return;
    }

    XrPlaneDetectorGetInfoEXT planeGetInfo{};
    planeGetInfo.type = XR_TYPE_PLANE_DETECTOR_GET_INFO_EXT;

```



```

planeGetInfo.time = time;
planeGetInfo.baseSpace = localSpace;

XrPlaneDetectorLocationsEXT planeLocations{};
planeLocations.type = XR_TYPE_PLANE_DETECTOR_LOCATIONS_EXT;
planeLocations.planeLocationCapacityInput = 0;
planeLocations.planeLocations = nullptr;

if (xrGetPlaneDetectionsEXT(planeDetector, &planeGetInfo, &planeLocations) !=
XR_SUCCESS ) {
    queryRunning = false;
    return;
}

if (planeLocations.planeLocationCountOutput > 0) {
    queryRunning = false;
    std::vector<XrPlaneDetectorLocationEXT>
        locationsBuffer(planeLocations.planeLocationCountOutput,
            { XR_TYPE_PLANE_DETECTOR_LOCATION_EXT });
    planeLocations.planeLocationCapacityInput =
        planeLocations.planeLocationCountOutput;
    planeLocations.planeLocations = locationsBuffer.data();

    CHK_XR(xrGetPlaneDetectionsEXT(planeDetector, &planeGetInfo,
&planeLocations));

    cachedPlaneLocations = locationsBuffer;

    for (int i = 0; i < planeLocations.planeLocationCountOutput; ++i) {
        const XrPosef& planeInLocalSpace = planeLocations.planeLocations[i].pose;
        auto planeId =
            planeLocations.planeLocations[i].planeId;
        auto polygonBufferCount =
            planeLocations.planeLocations[i].polygonBufferCount;

        for (uint32_t polygonBufferIndex=0; polygonBufferIndex <
polygonBufferCount; polygonBufferIndex++) {
            // polygonBufferIndex = 0 -> outside contour CCW
            // polygonBufferIndex > 0 -> holes CW
            XrPlaneDetectorPolygonBufferEXT polygonBuffer{};
            polygonBuffer.vertexCapacityInput = 0;

            CHK_XR(xrGetPlanePolygonBufferEXT(planeDetector,
                planeId, polygonBufferIndex, &polygonBuffer));

            // allocate space and use buffer
        }
        // plane planeInLocalSpace, planeType
    }
}

```

```

    }
}
};

while (1) {
    // ...
    // For every frame in frame loop
    // ...

    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    processPlanes(time);

    // Draw the planes as needed from cachedPlaneLocations.
    // drawPlanes(cachedPlaneLocations);

    // ...
    // Finish frame loop
    // ...
}

```

## New Object Types

- [XrPlaneDetectorEXT](#)

## New Enum Constants

[XrObjectType](#) enumeration is extended with:

- `XR_OBJECT_TYPE_PLANE_DETECTOR_EXT`

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_PLANE_DETECTOR_CREATE_INFO_EXT`
- `XR_TYPE_PLANE_DETECTOR_BEGIN_INFO_EXT`
- `XR_TYPE_PLANE_DETECTOR_GET_INFO_EXT`
- `XR_TYPE_PLANE_DETECTOR_LOCATION_EXT`
- `XR_TYPE_PLANE_DETECTOR_POLYGON_BUFFER_EXT`
- `XR_TYPE_SYSTEM_PLANE_DETECTION_PROPERTIES_EXT`

the [XrResult](#) enumeration is extended with:

- `XR_ERROR_SPACE_NOT_LOCATABLE_EXT`

- `XR_ERROR_PLANE_DETECTION_PERMISSION_DENIED_EXT`

### New Enums

- `XrPlaneDetectorOrientationEXT`
- `XrPlaneDetectorFlagsEXT`
- `XrPlaneDetectionStateEXT`
- `XrPlaneDetectionCapabilityFlagsEXT`
- `XrPlaneDetectorSemanticTypeEXT`

### New Structures

- `XrSystemPlaneDetectionPropertiesEXT`
- `XrPlaneDetectorCreateInfoEXT`
- `XrPlaneDetectorBeginInfoEXT`
- `XrPlaneDetectorGetInfoEXT`
- `XrPlaneDetectorLocationEXT`
- `XrPlaneDetectorPolygonBufferEXT`
- `XrExtent3DfEXT`

### New Functions

- `xrCreatePlaneDetectorEXT`
- `xrDestroyPlaneDetectorEXT`
- `xrBeginPlaneDetectionEXT`
- `xrGetPlaneDetectionStateEXT`
- `xrGetPlaneDetectionsEXT`
- `xrGetPlanePolygonBufferEXT`

### Version History

- Revision 1, 2023-06-26 (Ron Bessems)

## 12.36. XR\_EXT\_thermal\_query

### Name String

`XR_EXT_thermal_query`

### Extension Type

Instance extension

## Registered Extension Number

17

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-04-14

## IP Status

No known IP claims.

## Contributors

Armelle Laine, Qualcomm Technologies Inc, on behalf of Qualcomm Innovation Center, Inc

### 12.36.1. Overview

This extension provides an API to query a domain's current thermal warning level and current thermal trend.

### 12.36.2. Querying the current thermal level and trend

This query allows to determine the extent and urgency of the needed workload reduction and to verify that the mitigation measures efficiently reduce the temperature.

This query allows the application to retrieve the current `notificationLevel`, allowing to quickly verify whether the underlying system's thermal throttling is still in effect.

It also provides the application with the remaining temperature headroom (`tempHeadroom`) until thermal throttling occurs, and the current rate of change (`tempSlope`).

The most critical temperature of the domain is the one which is currently most likely to be relevant for thermal throttling.

To query the status of a given domain:

```
// Provided by XR_EXT_thermal_query
XrResult xrThermalGetTemperatureTrendEXT(
    XrSession                session,
    XrPerfSettingsDomainEXT  domain,
    XrPerfSettingsNotificationLevelEXT* notificationLevel,
    float*                   tempHeadroom,
    float*                   tempSlope);
```

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsDomainEXT {
    XR_PERF_SETTINGS_DOMAIN_CPU_EXT = 1,
    XR_PERF_SETTINGS_DOMAIN_GPU_EXT = 2,
    XR_PERF_SETTINGS_DOMAIN_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsDomainEXT;
```

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsNotificationLevelEXT {
    XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT = 0,
    XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT = 25,
    XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT = 75,
    XR_PERF_SETTINGS_NOTIFICATION_LEVEL_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsNotificationLevelEXT;
```

For the definition of the notification levels, see [Notification level definition](#).

### 12.36.3. Thermal Query API Reference

#### **xrThermalGetTemperatureTrendEXT**

```
// Provided by XR_EXT_thermal_query
XrResult xrThermalGetTemperatureTrendEXT(
    XrSession session,
    XrPerfSettingsDomainEXT domain,
    XrPerfSettingsNotificationLevelEXT* notificationLevel,
    float* tempHeadroom,
    float* tempSlope);
```

Allows to query the current temperature warning level of a domain, the remaining headroom and the trend.

## Parameter Descriptions

- `session` is a valid [XrSession](#) handle.
- `domain` : the processing domain
- `notificationLevel` : the current warning level
- `tempHeadroom` : temperature headroom in degrees Celsius, expressing how far the most-critical temperature of the domain is from its thermal throttling threshold temperature.
- `tempSlope` : the current trend in degrees Celsius per second of the most critical temperature of the domain.

## Valid Usage (Implicit)

- The [XR\\_EXT\\_thermal\\_query](#) extension **must** be enabled prior to calling [xrThermalGetTemperatureTrendEXT](#)
- `session` **must** be a valid [XrSession](#) handle
- `domain` **must** be a valid [XrPerfSettingsDomainEXT](#) value
- `notificationLevel` **must** be a pointer to an [XrPerfSettingsNotificationLevelEXT](#) value
- `tempHeadroom` **must** be a pointer to a `float` value
- `tempSlope` **must** be a pointer to a `float` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsDomainEXT {
    XR_PERF_SETTINGS_DOMAIN_CPU_EXT = 1,
    XR_PERF_SETTINGS_DOMAIN_GPU_EXT = 2,
    XR_PERF_SETTINGS_DOMAIN_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsDomainEXT;
```

```
// Provided by XR_EXT_performance_settings, XR_EXT_thermal_query
typedef enum XrPerfSettingsNotificationLevelEXT {
    XR_PERF_SETTINGS_NOTIF_LEVEL_NORMAL_EXT = 0,
    XR_PERF_SETTINGS_NOTIF_LEVEL_WARNING_EXT = 25,
    XR_PERF_SETTINGS_NOTIF_LEVEL_IMPAIRED_EXT = 75,
    XR_PERF_SETTINGS_NOTIFICATION_LEVEL_MAX_ENUM_EXT = 0x7FFFFFFF
} XrPerfSettingsNotificationLevelEXT;
```

### Version History

- Revision 1, 2017-11-30 (Armelle Laine)
- Revision 2, 2021-04-14 (Rylie Pavlik, Collabora, Ltd.)
  - Fix missing error code

## 12.37. XR\_EXT\_user\_presence

### Name String

`XR_EXT_user_presence`

### Extension Type

Instance extension

### Registered Extension Number

471

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-04-22

## IP Status

No known IP claims.

## Contributors

Yin Li, Microsoft

Bryce Hutchings, Microsoft

John Kearney, Meta Platforms

Andreas Loeve Selvik, Meta Platforms

Peter Kuhn, Unity Technologies

Jakob Bornecrantz, Collabora

### 12.37.1. Overview

This extension introduces a new event to notify when the system detected the change of user presence, such as when the user has taken off or put on an XR headset.

This event is typically used by an XR applications with non-XR experiences outside of the XR headset. For instance, some applications pause the game logic or video playback until the user puts on the headset, displaying an instructional message to the user in the mirror window on the desktop PC monitor. As another example, the application might use this event to disable a head-tracking driven avatar in an online meeting when the user has taken off the headset.

The user presence is fundamentally decoupled from the session lifecycle. Although the core spec for [XrSessionState](#) hinted potential correlation between the session state and user presence, in practice, such a connection **may** not consistently hold across various runtimes. Application **should** avoid relying on assumptions regarding these relationships between session state and user presence, instead, they should utilize this extension to reliably obtain user presence information.

### 12.37.2. System Supports User Presence

The [XrSystemUserPresencePropertiesEXT](#) structure is defined as:

```
// Provided by XR_EXT_user_presence
typedef struct XrSystemUserPresencePropertiesEXT {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsUserPresence;
} XrSystemUserPresencePropertiesEXT;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsUserPresence` is an [XrBool32](#) value that indicates whether the system supports user presence sensing.

The application **can** use the [XrSystemUserPresencePropertiesEXT](#) event in [xrGetSystemProperties](#) to detect if the given system supports the sensing of user presence.

If the system does not support user presence sensing, the runtime **must** return `XR_FALSE` for `supportsUserPresence` and **must** not queue the [XrEventDataUserPresenceChangedEXT](#) event for any session on this system.

In this case, an application typically assumes that the user is always present, as the runtime is unable to detect changes in user presence.

## Valid Usage (Implicit)

- The `XR_EXT_user_presence` extension **must** be enabled prior to using [XrSystemUserPresencePropertiesEXT](#)
- `type` **must** be `XR_TYPE_SYSTEM_USER_PRESENCE_PROPERTIES_EXT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.37.3. User Presence Changed Event

The [XrEventDataUserPresenceChangedEXT](#) structure is defined as:

```
// Provided by XR_EXT_user_presence
typedef struct XrEventDataUserPresenceChangedEXT {
    XrStructureType    type;
    const void*        next;
    XrSession          session;
    XrBool32           isUserPresent;
} XrEventDataUserPresenceChangedEXT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `session` is the [XrSession](#) that is receiving the notification.
- `isUserPresent` is an [XrBool32](#) value for new state of user presence after the change.

The [XrEventDataUserPresenceChangedEXT](#) event is queued for retrieval using [xrPollEvent](#) when the user presence is changed, as well as when a session starts running.

Receiving [XrEventDataUserPresenceChangedEXT](#) with the `isUserPresent` is `XR_TRUE` indicates that the system has detected the presence of a user in the XR experience. For example, this may indicate that the user has put on the headset, or has entered the tracking area of a non-head-worn XR system.

Receiving [XrEventDataUserPresenceChangedEXT](#) with the `isUserPresent` is `XR_FALSE` indicates that the system has detected the absence of a user in the XR experience. For example, this may indicate that the user has removed the headset or has stepped away from the tracking area of a non-head-worn XR system.

The runtime **must** queue this event upon a successful call to the [xrBeginSession](#) function, regardless of the value of `isUserPresent`, so that the application can be in sync on the state when a session begins running.

The runtime **must** return a valid [XrSession](#) handle for a [running session](#).

After the application calls [xrEndSession](#), a [running session](#) is ended and the runtime **must** not enqueue any more user presence events. Therefore, the application will no longer observe any changes of the `isUserPresent` until another [running session](#).

### Note



This extension does not require any specific correlation between user presence state and session state except that the [XrEventDataUserPresenceChangedEXT](#) event can not be observed without a running session. A runtime may choose to correlate the two states or keep them independent.

### Valid Usage (Implicit)

- The `XR_EXT_user_presence` extension **must** be enabled prior to using `XrEventDataUserPresenceChangedEXT`
- `type` **must** be `XR_TYPE_EVENT_DATA_USER_PRESENCE_CHANGED_EXT`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `session` **must** be a valid `XrSession` handle

## Example 2. Proper Method for Receiving OpenXR Event Data

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized

XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES};
XrSystemUserPresencePropertiesEXT userPresenceProperties
{XR_TYPE_SYSTEM_USER_PRESENCE_PROPERTIES_EXT};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
bool supportsUserPresence = userPresenceProperties.supportsUserPresence;

// When either the extension is not supported or the system does not support the
// sensor,
// the application typically assumes user always present, and initialize the
// isUserPresent
// to true before xrBeginSession and reset it to false after xrEndSession.
bool isUserPresent = true;

// Initialize an event buffer to hold the output.
XrEventDataBuffer event = {XR_TYPE_EVENT_DATA_BUFFER};
XrResult result = xrPollEvent(instance, &event);
if (result == XR_SUCCESS) {
    switch (event.type) {
        case XR_TYPE_EVENT_DATA_SESSION_STATE_CHANGED: {
            const XrEventDataSessionStateChanged& eventdata =
                *reinterpret_cast<XrEventDataSessionStateChanged*>(&event);
            XrSessionState sessionState = eventdata.state;
            switch(sessionState)
            {
                case XR_SESSION_STATE_READY: {
                    isUserPresent = true;
                    XrSessionBeginInfo beginInfo{XR_TYPE_SESSION_BEGIN_INFO};
                    CHK_XR(xrBeginSession(session, &beginInfo));
                    break;
                }
                case XR_SESSION_STATE_STOPPING:{
                    CHK_XR(xrEndSession(session));
                    isUserPresent = false;
                    break;
                }
            }
            break;
        }
        case XR_TYPE_EVENT_DATA_USER_PRESENCE_CHANGED_EXT: {
            const XrEventDataUserPresenceChangedEXT& eventdata =
                *reinterpret_cast<XrEventDataUserPresenceChangedEXT*>(&event);
```

```
        isUserPresent = eventdata.isUserPresent;
        // do_something(isUserPresent);
        break;
    }
}
```

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_EVENT_DATA_USER_PRESENCE_CHANGED_EXT`
- `XR_TYPE_SYSTEM_USER_PRESENCE_PROPERTIES_EXT`

## New Enums

## New Structures

- [XrSystemUserPresencePropertiesEXT](#)
- [XrEventDataUserPresenceChangedEXT](#)

## New Functions

## Issues

## Version History

- Revision 1, 2023-04-22 (Yin Li)
  - Initial extension description

# 12.38. XR\_EXT\_view\_configuration\_depth\_range

## Name String

`XR_EXT_view_configuration_depth_range`

## Extension Type

Instance extension

## Registered Extension Number

47

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2019-08-16

## IP Status

No known IP claims.

## Contributors

Blake Taylor, Magic Leap  
Gilles Cadet, Magic Leap  
Michael Liebenow, Magic Leap  
Supreet Suresh, Magic Leap  
Alex Turner, Microsoft  
Bryce Hutchings, Microsoft  
Yin Li, Microsoft

## Overview

For XR systems there may exist a per view recommended min/max depth range at which content should be rendered into the virtual world. The depth range may be driven by several factors, including user comfort, or fundamental capabilities of the system.

Displaying rendered content outside the recommended min/max depth range would violate the system requirements for a properly integrated application, and can result in a poor user experience due to observed visual artifacts, visual discomfort, or fatigue. The near/far depth values will fall in the range of  $(0..+\infty]$  where  $\max(\text{recommendedNearZ}, \text{minNearZ}) < \min(\text{recommendedFarZ}, \text{maxFarZ})$ . Infinity is defined matching the standard library definition such that `std::isinf` will return true for a returned infinite value.

In order to provide the application with the appropriate depth range at which to render content for each [XrViewConfigurationView](#), this extension provides additional view configuration information, as defined by [XrViewConfigurationDepthRangeEXT](#), to inform the application of the min/max recommended and absolute distances at which content should be rendered for that view.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_VIEW_CONFIGURATION_DEPTH_RANGE_EXT`

## New Enums

## New Structures

The `XrViewConfigurationDepthRangeEXT` structure is defined as:

```
// Provided by XR_EXT_view_configuration_depth_range
typedef struct XrViewConfigurationDepthRangeEXT {
    XrStructureType    type;
    void*              next;
    float               recommendedNearZ;
    float               minNearZ;
    float               recommendedFarZ;
    float               maxFarZ;
} XrViewConfigurationDepthRangeEXT;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `recommendedNearZ` is the recommended minimum positive distance in meters that content should be rendered for the view to achieve the best user experience.
- `minNearZ` is the absolute minimum positive distance in meters that content should be rendered for the view.
- `recommendedFarZ` is the recommended maximum positive distance in meters that content should be rendered for the view to achieve the best user experience.
- `maxFarZ` is the absolute maximum positive distance in meters that content should be rendered for the view.

When enumerating the view configurations with `xrEnumerateViewConfigurationViews`, the application **can** provide a pointer to an `XrViewConfigurationDepthRangeEXT` in the `next` chain of `XrViewConfigurationView`.

## Valid Usage (Implicit)

- The [XR\\_EXT\\_view\\_configuration\\_depth\\_range](#) extension **must** be enabled prior to using [XrViewConfigurationDepthRangeEXT](#)
- **type** **must** be [XR\\_TYPE\\_VIEW\\_CONFIGURATION\\_DEPTH\\_RANGE\\_EXT](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

### New Functions

### Issues

### Version History

- Revision 1, 2019-10-01 (Blake Taylor)
  - Initial proposal.

## 12.39. XR\_EXT\_win32\_appcontainer\_compatible

### Name String

[XR\\_EXT\\_win32\\_appcontainer\\_compatible](#)

### Extension Type

Instance extension

### Registered Extension Number

58

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2019-12-16

### IP Status

No known IP claims.

### Contributors

Yin Li, Microsoft

Alex Turner, Microsoft

Lachlan Ford, Microsoft



## Overview

To minimize opportunities for malicious manipulation, a common practice on the Windows OS is to isolate the application process in an [AppContainer execution environment](#). In order for a runtime to work properly in such an application process, the runtime **must** properly [set ACL to device resources and cross process resources](#).

An application running in an AppContainer process **can** request for a runtime to enable such AppContainer compatibility by adding `XR_EXT_WIN32_APPCONTAINER_COMPATIBLE_EXTENSION_NAME` to `enabledExtensionNames` of `XrInstanceCreateInfo` when calling `xrCreateInstance`. If the runtime is not capable of running properly within the AppContainer execution environment, it **must** return `XR_ERROR_EXTENSION_NOT_PRESENT`.

If the runtime supports this extension, it **can** further inspect the capability based on the connected device. If the XR system cannot support an AppContainer execution environment, the runtime **must** return `XR_ERROR_FORM_FACTOR_UNAVAILABLE` when the application calls `xrGetSystem`.

If the call to `xrGetSystem` successfully returned with a valid `XrSystemId`, the application **can** rely on the runtime working properly in the AppContainer execution environment.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2019-12-16 (Yin Li)
  - Initial proposal.

# 12.40. XR\_ALMALENCE\_digital\_lens\_control

## Name String

`XR_ALMALENCE_digital_lens_control`

## Extension Type

Instance extension

## Registered Extension Number

197

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-11-08

## IP Status

No known IP claims.

## Contributors

Ivan Chupakhin, Almalence Inc.

Dmitry Shmunk, Almalence Inc.

## Overview

Digital Lens for VR (DLVR) is a computational lens aberration correction technology enabling high resolution, visual clarity and fidelity in VR head mounted displays. The Digital Lens allows to overcome two fundamental factors limiting VR picture quality, size constraints and presence of a moving optical element — the eye pupil.

### Features:

- Complete removal of lateral chromatic aberrations, across the entire FoV, at all gaze directions.
- Correction of longitudinal chromatic aberrations, lens blur and higher order aberrations.
- Increase of visible resolution.
- Enhancement of edge contrast (otherwise degraded due to lens smear).
- Enables high quality at wide FoV.

For OpenXR runtimes DLVR is implemented as implicit API Layer distributed by Almalence Inc. as installable package. DLVR utilize eye tracking data (eye pupil coordinates and gaze direction) to produce corrections of render frames. As long as current core OpenXR API does not expose an eye tracking data, DLVR API Layer relies on 3rd-party eye tracking runtimes.

### List of supported eye tracking devices:

- *Tobii\_VR4\_CARBON\_P1* (HP Reverb G2 Omnicept Edition)
- *Tobii\_VR4\_U2\_P2* (HTC Vive Pro Eye)

This extension enables the handling of the Digital Lens for VR API Layer by calling [xrSetDigitalLensControlALMALENCE](#).

## New Object Types

## New Flag Types

```
typedef XrFlags64 XrDigitalLensControlFlagsALMALENCE;
```

```
// Flag bits for XrDigitalLensControlFlagsALMALENCE
static const XrDigitalLensControlFlagsALMALENCE
XR_DIGITAL_LENS_CONTROL_PROCESSING_DISABLE_BIT_ALMALENCE = 0x00000001;
```

### Flag Descriptions

- `XR_DIGITAL_LENS_CONTROL_PROCESSING_DISABLE_BIT_ALMALENCE` — disables Digital Lens processing of render textures

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_DIGITAL_LENS_CONTROL_ALMALENCE`

## New Enums

## New Structures

The [XrDigitalLensControlALMALENCE](#) structure is defined as:

```
typedef struct XrDigitalLensControlALMALENCE {
    XrStructureType          type;
    const void*              next;
    XrDigitalLensControlFlagsALMALENCE flags;
} XrDigitalLensControlALMALENCE;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` must be `NULL`. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of [XrDigitalLensControlFlagBitsALMALENCE](#) indicating various characteristics desired for the Digital Lens.

## Valid Usage (Implicit)

- The [XR\\_ALMALENCE\\_digital\\_lens\\_control](#) extension **must** be enabled prior to using [XrDigitalLensControlALMALENCE](#)
- `type` **must** be `XR_TYPE_DIGITAL_LENS_CONTROL_ALMALENCE`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be a valid combination of [XrDigitalLensControlFlagBitsALMALENCE](#) values
- `flags` **must** not be `0`

## New Functions

The [xrSetDigitalLensControlALMALENCE](#) function is defined as:

```
// Provided by XR_ALMALENCE_digital_lens_control
XrResult xrSetDigitalLensControlALMALENCE(
    XrSession session,
    const XrDigitalLensControlALMALENCE* digitalLensControl);
```

## Parameter Descriptions

- `session` is a handle to a running [XrSession](#).
- `digitalLensControl` is the [XrDigitalLensControlALMALENCE](#) that contains desired characteristics for the Digital Lens

[xrSetDigitalLensControlALMALENCE](#) handles state of Digital Lens API Layer

## Valid Usage (Implicit)

- The `XR_ALMALENCE_digital_lens_control` extension **must** be enabled prior to calling `xrSetDigitalLensControlALMALENCE`
- `session` **must** be a valid `XrSession` handle
- `digitalLensControl` **must** be a pointer to a valid `XrDigitalLensControlALMALENCE` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

## Issues

## Version History

- Revision 1, 2021-11-08 (Ivan Chupakhin)
  - Initial draft

# 12.41. XR\_EPIC\_view\_configuration\_fov

## Name String

`XR_EPIC_view_configuration_fov`

## Extension Type

Instance extension

## Registered Extension Number

60

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2020-03-05

## IP Status

No known IP claims.

## Contributors

Jules Blok, Epic Games

## Overview

This extension allows the application to retrieve the recommended and maximum field-of-view using [xrEnumerateViewConfigurationViews](#). These field-of-view parameters can be used during initialization of the application before creating a session.

The field-of-view given here **should** not be used for rendering, see [xrLocateViews](#) to retrieve the field-of-view for rendering.

For views with `fovMutable` set to `XR_TRUE` the maximum field-of-view **should** specify the upper limit that runtime can support. If the view has `fovMutable` set to `XR_FALSE` the runtime **must** set `maxMutableFov` to be the same as `recommendedFov`.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

The [XrViewConfigurationViewFovEPIC](#) structure is an output struct which can be added to the next chain of [XrViewConfigurationView](#) to retrieve the field-of-view for that view.

```
// Provided by XR_EPIC_view_configuration_fov
typedef struct XrViewConfigurationViewFovEPIC {
    XrStructureType    type;
    const void*        next;
    XrFovf              recommendedFov;
    XrFovf              maxMutableFov;
} XrViewConfigurationViewFovEPIC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `recommendedFov` is the recommended field-of-view based on the current user IPD.
- `maxMutableFov` is the maximum field-of-view that the runtime can display.

## Valid Usage (Implicit)

- The [XR\\_EPIC\\_view\\_configuration\\_fov](#) extension **must** be enabled prior to using [XrViewConfigurationViewFovEPIC](#)
- `type` **must** be `XR_TYPE_VIEW_CONFIGURATION_VIEW_FOV_EPIC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

## Issues

## Version History

- Revision 2, 2020-06-04 (Jules Blok)
  - Fixed incorrect member name.
- Revision 1, 2020-03-05 (Jules Blok)
  - Initial version.

# 12.42. XR\_FB\_android\_surface\_swapchain\_create

## Name String

`XR_FB_android_surface_swapchain_create`

## Extension Type

Instance extension

## Registered Extension Number

71

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_KHR\\_android\\_surface\\_swapchain](#)

## Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

Tomislav Novak, Facebook

## Overview

This extension provides support for the specification of Android Surface specific swapchain create flags.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

These additional create flags are specified by attaching a [XrAndroidSurfaceSwapchainCreateInfoFB](#) structure to the [next](#) chain of an [XrSwapchainCreateInfo](#) structure.

## New Object Types

## New Flag Types

```
typedef XrFlags64 XrAndroidSurfaceSwapchainFlagsFB;
```



```
// Flag bits for XrAndroidSurfaceSwapchainFlagsFB
static const XrAndroidSurfaceSwapchainFlagsFB
XR_ANDROID_SURFACE_SWAPCHAIN_SYNCHRONOUS_BIT_FB = 0x00000001;
static const XrAndroidSurfaceSwapchainFlagsFB
XR_ANDROID_SURFACE_SWAPCHAIN_USE_TIMESTAMPS_BIT_FB = 0x00000002;
```

## Flag Descriptions

- `XR_ANDROID_SURFACE_SWAPCHAIN_SYNCHRONOUS_BIT_FB` indicates the underlying BufferQueue should be created in synchronous mode, allowing multiple buffers to be queued instead of always replacing the last buffer. Buffers are retired in order, and the producer may block until a new buffer is available.
- `XR_ANDROID_SURFACE_SWAPCHAIN_USE_TIMESTAMPS_BIT_FB` indicates the compositor should acquire the most recent buffer whose presentation timestamp is not greater than the expected display time of the final composited frame.

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_ANDROID_SURFACE_SWAPCHAIN_CREATE_INFO_FB`

## New Enums

- `XR_ANDROID_SURFACE_SWAPCHAIN_SYNCHRONOUS_BIT_FB`
- `XR_ANDROID_SURFACE_SWAPCHAIN_USE_TIMESTAMPS_BIT_FB`

## New Structures

The `XrAndroidSurfaceSwapchainCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_android_surface_swapchain_create
typedef struct XrAndroidSurfaceSwapchainCreateInfoFB {
    XrStructureType          type;
    const void*              next;
    XrAndroidSurfaceSwapchainFlagsFB createFlags;
} XrAndroidSurfaceSwapchainCreateInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `createFlags` is `0` or one or more [XrAndroidSurfaceSwapchainFlagBitsFB](#) which indicate various characteristics desired for the Android Surface Swapchain.

[XrAndroidSurfaceSwapchainCreateInfoFB](#) contains additional Android Surface specific create flags when calling [xrCreateSwapchainAndroidSurfaceKHR](#). The [XrAndroidSurfaceSwapchainCreateInfoFB](#) structure **must** be provided in the `next` chain of the [XrSwapchainCreateInfo](#) structure when calling [xrCreateSwapchainAndroidSurfaceKHR](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_android\\_surface\\_swapchain\\_create](#) extension **must** be enabled prior to using [XrAndroidSurfaceSwapchainCreateInfoFB](#)
- `type` **must** be [XR\\_TYPE\\_ANDROID\\_SURFACE\\_SWAPCHAIN\\_CREATE\\_INFO\\_FB](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `createFlags` **must** be a valid combination of [XrAndroidSurfaceSwapchainFlagBitsFB](#) values
- `createFlags` **must** not be `0`

## New Functions

## Issues

## Version History

- Revision 1, 2020-12-10 (Gloria Kennickell)
  - Initial draft

# 12.43. XR\_FB\_body\_tracking

## Name String

[XR\\_FB\\_body\\_tracking](#)

## Extension Type

Instance extension

## Registered Extension Number

77

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-07-18

## IP Status

No known IP claims.

## Contributors

Giancarlo Di Biase, Meta

Dikpal Reddy, Meta

Igor Tceglevskii, Meta

### 12.43.1. Overview

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.

### 12.43.2. Inspect system capability

An application **can** inspect whether the system is capable of body tracking by extending the [XrSystemProperties](#) with [XrSystemBodyTrackingPropertiesFB](#) structure when calling [xrGetSystemProperties](#).

```
// Provided by XR_FB_body_tracking
typedef struct XrSystemBodyTrackingPropertiesFB {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsBodyTracking;
} XrSystemBodyTrackingPropertiesFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsBodyTracking` is an [XrBool32](#), indicating if current system is capable of receiving body tracking input.

If a runtime returns `XR_FALSE` for `supportsBodyTracking`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateBodyTrackerFB](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_body\\_tracking](#) extension **must** be enabled prior to using [XrSystemBodyTrackingPropertiesFB](#)
- `type` **must** be `XR_TYPE_SYSTEM_BODY_TRACKING_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.43.3. Create a body tracker handle

The [XrBodyTrackerFB](#) handle represents the resources for body tracking.

```
// Provided by XR_FB_body_tracking
XR_DEFINE_HANDLE(XrBodyTrackerFB)
```

This handle **can** be used to locate body joints using [xrLocateBodyJointsFB](#) function.

A body tracker provides joint locations with an unobstructed range of human body motion.

It also provides the estimated scale of this body.

An application **can** create an [XrBodyTrackerFB](#) handle using [xrCreateBodyTrackerFB](#) function.

```
// Provided by XR_FB_body_tracking
XrResult xrCreateBodyTrackerFB(
    XrSession session,
    const XrBodyTrackerCreateInfoFB* createInfo,
    XrBodyTrackerFB* bodyTracker);
```

## Parameter Descriptions

- `session` is an `XrSession` in which the body tracker will be active.
- `createInfo` is the `XrBodyTrackerCreateInfoFB` used to specify the body tracker.
- `bodyTracker` is the returned `XrBodyTrackerFB` handle.

If the system does not support body tracking, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateBodyTrackerFB`. In this case, the runtime **must** return `XR_FALSE` for `XrSystemBodyTrackingPropertiesFB::supportsBodyTracking` when the function `xrGetSystemProperties` is called, so that the application **can** avoid creating a body tracker.

## Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to calling `xrCreateBodyTrackerFB`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrBodyTrackerCreateInfoFB` structure
- `bodyTracker` **must** be a pointer to an `XrBodyTrackerFB` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrBodyTrackerCreateInfoFB` structure describes the information to create an `XrBodyTrackerFB` handle.

```
// Provided by XR_FB_body_tracking
typedef struct XrBodyTrackerCreateInfoFB {
    XrStructureType    type;
    const void*        next;
    XrBodyJointSetFB  bodyJointSet;
} XrBodyTrackerCreateInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `bodyJointSet` is an `XrBodyJointSetFB` that describes the set of body joints to retrieve.

## Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to using `XrBodyTrackerCreateInfoFB`
- `type` **must** be `XR_TYPE_BODY_TRACKER_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `bodyJointSet` **must** be a valid `XrBodyJointSetFB` value

The `XrBodyJointSetFB` enum describes the set of body joints to track when creating an `XrBodyTrackerFB`.

```
// Provided by XR_FB_body_tracking
typedef enum XrBodyJointSetFB {
    XR_BODY_JOINT_SET_DEFAULT_FB = 0,
    XR_BODY_JOINT_SET_MAX_ENUM_FB = 0x7FFFFFFF
} XrBodyJointSetFB;
```

## Enumerant Descriptions

- `XR_BODY_JOINT_SET_DEFAULT_FB` — Indicates that the created `XrBodyTrackerFB` tracks the set of body joints described by `XrBodyJointFB` enum, i.e. the `xrLocateBodyJointsFB` function returns an array of joint locations with the count of `XR_BODY_JOINT_COUNT_FB` and can be indexed using `XrBodyJointFB`.

`xrDestroyBodyTrackerFB` function releases the `bodyTracker` and the underlying resources when the body tracking experience is over.

```
// Provided by XR_FB_body_tracking
XrResult xrDestroyBodyTrackerFB(
    XrBodyTrackerFB          bodyTracker);
```

## Parameter Descriptions

- `bodyTracker` is an `XrBodyTrackerFB` previously created by `xrCreateBodyTrackerFB`.

## Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to calling `xrDestroyBodyTrackerFB`
- `bodyTracker` **must** be a valid `XrBodyTrackerFB` handle

## Thread Safety

- Access to `bodyTracker`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## 12.43.4. Locate body joints

The `xrLocateBodyJointsFB` function locates an array of body joints to a base space at a given time.

```
// Provided by XR_FB_body_tracking
XrResult xrLocateBodyJointsFB(
    XrBodyTrackerFB                bodyTracker,
    const XrBodyJointsLocateInfoFB* locateInfo,
    XrBodyJointLocationsFB*        locations);
```

## Parameter Descriptions

- `bodyTracker` is an `XrBodyTrackerFB` previously created by `xrCreateBodyTrackerFB`.
- `locateInfo` is a pointer to `XrBodyJointsLocateInfoFB` describing information to locate body joints.
- `locations` is a pointer to `XrBodyJointLocationsFB` receiving the returned body joint locations.



## Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to calling `xrLocateBodyJointsFB`
- `bodyTracker` **must** be a valid `XrBodyTrackerFB` handle
- `locateInfo` **must** be a pointer to a valid `XrBodyJointsLocateInfoFB` structure
- `locations` **must** be a pointer to an `XrBodyJointLocationsFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrBodyJointsLocateInfoFB` structure describes the information to locate body joints.

```
// Provided by XR_FB_body_tracking
typedef struct XrBodyJointsLocateInfoFB {
    XrStructureType    type;
    const void*       next;
    XrSpace            baseSpace;
    XrTime             time;
} XrBodyJointsLocateInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `baseSpace` is an [XrSpace](#) within which the returned body joint locations will be represented.
- `time` is an [XrTime](#) at which to locate the body joints.

Callers **should** request a time equal to the predicted display time for the rendered frame. The system will employ appropriate modeling to support body tracking at this time.

## Valid Usage (Implicit)

- The [XR\\_FB\\_body\\_tracking](#) extension **must** be enabled prior to using [XrBodyJointsLocateInfoFB](#)
- `type` **must** be `XR_TYPE_BODY_JOINTS_LOCATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `baseSpace` **must** be a valid [XrSpace](#) handle

[XrBodyJointLocationsFB](#) structure returns the state of the body joint locations.

```
// Provided by XR_FB_body_tracking
typedef struct XrBodyJointLocationsFB {
    XrStructureType      type;
    void*                next;
    XrBool32             isActive;
    float                confidence;
    uint32_t             jointCount;
    XrBodyJointLocationFB* jointLocations;
    uint32_t             skeletonChangedCount;
    XrTime               time;
} XrBodyJointLocationsFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `isActive` is an `XrBool32` indicating if the body tracker is actively tracking.
- `confidence` is a `float` between 0 and 1 which represents the confidence for the returned body pose. A value of 0 means there is no confidence in the pose returned, and a value of 1 means maximum confidence in the returned body pose.
- `jointCount` is a `uint32_t` describing the count of elements in `jointLocations` array.
- `jointLocations` is an application-allocated array of `XrBodyJointLocationFB` that will be filled with joint locations.
- `skeletonChangedCount` is an output `uint32_t` incremental counter indicating that the skeleton scale proportions have changed. `xrGetBodySkeletonFB` can be called when this counter increases to get the latest body proportions/scale.
- `time` is an `XrTime` time at which the returned joints are tracked. Equals the time at which the joints were requested if the interpolation at the time was successful.

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `jointCount` does not equal to the number of joints defined by the `XrBodyJointSetFB` used to create the `XrBodyTrackerFB`.

The runtime **must** return `jointLocations` representing the range of human body motion, without any obstructions. Input systems that either obstruct the movement of the user's body (for example, a held controller preventing the user from making a fist) or input systems that have only limited ability to track finger positions **must** use the information available to them to emulate an unobstructed range of motion.

The runtime **must** update the `jointLocations` array ordered so that it is indexed using the corresponding body joint enum (e.g. `XrBodyJointFB`) as described by `XrBodyJointSetFB` when creating the `XrBodyTrackerFB`. For example, when the `XrBodyTrackerFB` is created with `XR_BODY_JOINT_SET_DEFAULT_FB`, the application **must** set the `jointCount` to `XR_BODY_JOINT_COUNT_FB`, and the runtime **must** fill the `jointLocations` array ordered so that it is indexed by the `XrBodyJointFB` enum.

If the returned `isActive` is true, the runtime **must** return all joint locations with both `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` set. However, in this case, some joint space locations **may** be untracked (i.e. `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` or `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` is unset).

If the returned `isActive` is false, it indicates that the body tracker did not detect the body input, the application lost input focus, or the consent for body tracking was denied by the user. In this case, the runtime **must** return all `jointLocations` with neither `XR_SPACE_LOCATION_POSITION_VALID_BIT` nor

`XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` set.

### Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to using `XrBodyJointLocationsFB`
- `type` **must** be `XR_TYPE_BODY_JOINT_LOCATIONS_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `jointLocations` **must** be a pointer to an array of `jointCount` `XrBodyJointLocationFB` structures
- The `jointCount` parameter **must** be greater than `0`

`XrBodyJointLocationFB` structure describes the position, orientation, and radius of a body joint.

```
// Provided by XR_FB_body_tracking
typedef struct XrBodyJointLocationFB {
    XrSpaceLocationFlags    locationFlags;
    XrPosef                 pose;
} XrBodyJointLocationFB;
```

### Member Descriptions

- `locationFlags` is a bitfield, with bit masks defined in `XrSpaceLocationFlagBits`, to indicate which members contain valid data. If none of the bits are set, no other fields in this structure **should** be considered to be valid or meaningful.
- `pose` is an `XrPosef` defining the position and orientation of the origin of a body joint within the reference frame of the corresponding `XrBodyJointsLocateInfoFB::baseSpace`.

### Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to using `XrBodyJointLocationFB`
- `locationFlags` **must** be a valid combination of `XrSpaceLocationFlagBits` values
- `locationFlags` **must** not be `0`

## 12.43.5. Retrieve body skeleton

The `xrGetBodySkeletonFB` function returns the body skeleton in T-pose.

```
// Provided by XR_FB_body_tracking
XrResult xrGetBodySkeletonFB(
    XrBodyTrackerFB          bodyTracker,
    XrBodySkeletonFB*       skeleton);
```

## Parameter Descriptions

- `bodyTracker` is an `XrBodyTrackerFB` previously created by `xrCreateBodyTrackerFB`.
- `skeleton` is a pointer to `XrBodySkeletonFB` receiving the returned body skeleton hierarchy.

This function **can** be used to query the skeleton scale and proportions in conjunction with `XrBodyJointLocationsFB::skeletonChangedCount`. `XrBodyJointLocationsFB::skeletonChangedCount` is incremented whenever the tracking auto-calibrates the user skeleton scale and proportions.

## Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to calling `xrGetBodySkeletonFB`
- `bodyTracker` **must** be a valid `XrBodyTrackerFB` handle
- `skeleton` **must** be a pointer to an `XrBodySkeletonFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `XrBodySkeletonFB` structure is a container to represent the body skeleton in T-pose including the joint hierarchy.

```
// Provided by XR_FB_body_tracking
typedef struct XrBodySkeletonFB {
    XrStructureType      type;
    void*                next;
    uint32_t             jointCount;
    XrBodySkeletonJointFB* joints;
} XrBodySkeletonFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `jointCount` is an `uint32_t` describing the count of elements in `joints` array.
- `joints` is an application-allocated array of [XrBodySkeletonJointFB](#) that will be filled with skeleton joint elements.

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `jointCount` does not equal to the number of joints defined by the [XrBodyJointSetFB](#) used to create the [XrBodyTrackerFB](#).

The runtime **must** return `joints` representing the default pose of the current estimation regarding the user's skeleton.

## Valid Usage (Implicit)

- The `XR_FB_body_tracking` extension **must** be enabled prior to using [XrBodySkeletonFB](#)
- `type` **must** be `XR_TYPE_BODY_SKELETON_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `joints` **must** be a pointer to an array of `jointCount` [XrBodySkeletonJointFB](#) structures
- The `jointCount` parameter **must** be greater than `0`

[XrBodySkeletonJointFB](#) structure describes the position, orientation of the joint in space, and position of the joint in the skeleton hierarchy.

```
// Provided by XR_FB_body_tracking
typedef struct XrBodySkeletonJointFB {
    int32_t    joint;
    int32_t    parentJoint;
    XrPosef    pose;
} XrBodySkeletonJointFB;
```

## Member Descriptions

- `joint` is an index of a joint using the corresponding body joint enum (e.g. [XrBodyJointFB](#)).
- `parentJoint` is an index of a parent joint of that joint, using the corresponding body joint enum (e.g. [XrBodyJointFB](#)).
- `pose` is an [XrPosef](#) defining the position and orientation of the origin of a body joint within the reference frame of the corresponding [XrBodyJointsLocateInfoFB::baseSpace](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_body\\_tracking](#) extension **must** be enabled prior to using [XrBodySkeletonJointFB](#)

### 12.43.6. Example code for locating body joints

The following example code demonstrates how to locate all body joints relatively to a base space.

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized
XrSpace baseSpace; // previously initialized, e.g. from
                    // XR_REFERENCE_SPACE_TYPE_LOCAL

// Inspect body tracking system properties
XrSystemBodyTrackingPropertiesFB bodyTrackingSystemProperties{
    XR_TYPE_SYSTEM_BODY_TRACKING_PROPERTIES_FB};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES,
    &bodyTrackingSystemProperties};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
if (!bodyTrackingSystemProperties.supportsBodyTracking) {
    // The system does not support body tracking
    return;
}

// Get function pointer for xrCreateBodyTrackerFB
```

```

PFN_xrCreateBodyTrackerFB pfnCreateBodyTrackerFB;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateBodyTrackerFB",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnCreateBodyTrackerFB)));

// Create a body tracker that tracks default set of body joints.
XrBodyTrackerFB bodyTracker = {};
{
    XrBodyTrackerCreateInfoFB createInfo{XR_TYPE_BODY_TRACKER_CREATE_INFO_FB};
    createInfo.bodyJointSet = XR_BODY_JOINT_SET_DEFAULT_FB;
    CHK_XR(pfnCreateBodyTrackerFB(session, &createInfo, &bodyTracker));
}

// Allocate buffers to receive joint location data before frame
// loop starts.
XrBodyJointLocationFB jointLocations[XR_BODY_JOINT_COUNT_FB];
XrBodyJointLocationsFB locations{XR_TYPE_BODY_JOINT_LOCATIONS_FB};
locations.jointCount = XR_BODY_JOINT_COUNT_FB;
locations.jointLocations = jointLocations;

// Get function pointer for xrLocateBodyJointsFB.
PFN_xrLocateBodyJointsFB pfnLocateBodyJointsFB;
CHK_XR(xrGetInstanceProcAddr(instance, "xrLocateBodyJointsFB",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnLocateBodyJointsFB)));
while (1) {
    // ...
    // For every frame in the frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrBodyJointsLocateInfoFB locateInfo{XR_TYPE_BODY_JOINTS_LOCATE_INFO_FB};
    locateInfo.baseSpace = baseSpace;
    locateInfo.time = time;

    CHK_XR(pfnLocateBodyJointsFB(bodyTracker, &locateInfo, &locations));

    if (locations.isActive) {
        // The returned joint location array is directly indexed with
        // XrBodyJointFB enum.
        const XrPosef &indexTip =
            jointLocations[XR_BODY_JOINT_LEFT_HAND_INDEX_TIP_FB].pose;
    }
}

```



## 12.43.7. Conventions of body joints

This extension defines 70 joints for body tracking: 18 core body joints + 52 hand joints.

```
// Provided by XR_FB_body_tracking
typedef enum XrBodyJointFB {
    XR_BODY_JOINT_ROOT_FB = 0,
    XR_BODY_JOINT_HIPS_FB = 1,
    XR_BODY_JOINT_SPINE_LOWER_FB = 2,
    XR_BODY_JOINT_SPINE_MIDDLE_FB = 3,
    XR_BODY_JOINT_SPINE_UPPER_FB = 4,
    XR_BODY_JOINT_CHEST_FB = 5,
    XR_BODY_JOINT_NECK_FB = 6,
    XR_BODY_JOINT_HEAD_FB = 7,
    XR_BODY_JOINT_LEFT_SHOULDER_FB = 8,
    XR_BODY_JOINT_LEFT_SCAPULA_FB = 9,
    XR_BODY_JOINT_LEFT_ARM_UPPER_FB = 10,
    XR_BODY_JOINT_LEFT_ARM_LOWER_FB = 11,
    XR_BODY_JOINT_LEFT_HAND_WRIST_TWIST_FB = 12,
    XR_BODY_JOINT_RIGHT_SHOULDER_FB = 13,
    XR_BODY_JOINT_RIGHT_SCAPULA_FB = 14,
    XR_BODY_JOINT_RIGHT_ARM_UPPER_FB = 15,
    XR_BODY_JOINT_RIGHT_ARM_LOWER_FB = 16,
    XR_BODY_JOINT_RIGHT_HAND_WRIST_TWIST_FB = 17,
    XR_BODY_JOINT_LEFT_HAND_PALM_FB = 18,
    XR_BODY_JOINT_LEFT_HAND_WRIST_FB = 19,
    XR_BODY_JOINT_LEFT_HAND_THUMB_METACARPAL_FB = 20,
    XR_BODY_JOINT_LEFT_HAND_THUMB_PROXIMAL_FB = 21,
    XR_BODY_JOINT_LEFT_HAND_THUMB_DISTAL_FB = 22,
    XR_BODY_JOINT_LEFT_HAND_THUMB_TIP_FB = 23,
    XR_BODY_JOINT_LEFT_HAND_INDEX_METACARPAL_FB = 24,
    XR_BODY_JOINT_LEFT_HAND_INDEX_PROXIMAL_FB = 25,
    XR_BODY_JOINT_LEFT_HAND_INDEX_INTERMEDIATE_FB = 26,
    XR_BODY_JOINT_LEFT_HAND_INDEX_DISTAL_FB = 27,
    XR_BODY_JOINT_LEFT_HAND_INDEX_TIP_FB = 28,
    XR_BODY_JOINT_LEFT_HAND_MIDDLE_METACARPAL_FB = 29,
    XR_BODY_JOINT_LEFT_HAND_MIDDLE_PROXIMAL_FB = 30,
    XR_BODY_JOINT_LEFT_HAND_MIDDLE_INTERMEDIATE_FB = 31,
    XR_BODY_JOINT_LEFT_HAND_MIDDLE_DISTAL_FB = 32,
    XR_BODY_JOINT_LEFT_HAND_MIDDLE_TIP_FB = 33,
    XR_BODY_JOINT_LEFT_HAND_RING_METACARPAL_FB = 34,
    XR_BODY_JOINT_LEFT_HAND_RING_PROXIMAL_FB = 35,
    XR_BODY_JOINT_LEFT_HAND_RING_INTERMEDIATE_FB = 36,
    XR_BODY_JOINT_LEFT_HAND_RING_DISTAL_FB = 37,
    XR_BODY_JOINT_LEFT_HAND_RING_TIP_FB = 38,
```

```

XR_BODY_JOINT_LEFT_HAND_LITTLE_METACARPAL_FB = 39,
XR_BODY_JOINT_LEFT_HAND_LITTLE_PROXIMAL_FB = 40,
XR_BODY_JOINT_LEFT_HAND_LITTLE_INTERMEDIATE_FB = 41,
XR_BODY_JOINT_LEFT_HAND_LITTLE_DISTAL_FB = 42,
XR_BODY_JOINT_LEFT_HAND_LITTLE_TIP_FB = 43,
XR_BODY_JOINT_RIGHT_HAND_PALM_FB = 44,
XR_BODY_JOINT_RIGHT_HAND_WRIST_FB = 45,
XR_BODY_JOINT_RIGHT_HAND_THUMB_METACARPAL_FB = 46,
XR_BODY_JOINT_RIGHT_HAND_THUMB_PROXIMAL_FB = 47,
XR_BODY_JOINT_RIGHT_HAND_THUMB_DISTAL_FB = 48,
XR_BODY_JOINT_RIGHT_HAND_THUMB_TIP_FB = 49,
XR_BODY_JOINT_RIGHT_HAND_INDEX_METACARPAL_FB = 50,
XR_BODY_JOINT_RIGHT_HAND_INDEX_PROXIMAL_FB = 51,
XR_BODY_JOINT_RIGHT_HAND_INDEX_INTERMEDIATE_FB = 52,
XR_BODY_JOINT_RIGHT_HAND_INDEX_DISTAL_FB = 53,
XR_BODY_JOINT_RIGHT_HAND_INDEX_TIP_FB = 54,
XR_BODY_JOINT_RIGHT_HAND_MIDDLE_METACARPAL_FB = 55,
XR_BODY_JOINT_RIGHT_HAND_MIDDLE_PROXIMAL_FB = 56,
XR_BODY_JOINT_RIGHT_HAND_MIDDLE_INTERMEDIATE_FB = 57,
XR_BODY_JOINT_RIGHT_HAND_MIDDLE_DISTAL_FB = 58,
XR_BODY_JOINT_RIGHT_HAND_MIDDLE_TIP_FB = 59,
XR_BODY_JOINT_RIGHT_HAND_RING_METACARPAL_FB = 60,
XR_BODY_JOINT_RIGHT_HAND_RING_PROXIMAL_FB = 61,
XR_BODY_JOINT_RIGHT_HAND_RING_INTERMEDIATE_FB = 62,
XR_BODY_JOINT_RIGHT_HAND_RING_DISTAL_FB = 63,
XR_BODY_JOINT_RIGHT_HAND_RING_TIP_FB = 64,
XR_BODY_JOINT_RIGHT_HAND_LITTLE_METACARPAL_FB = 65,
XR_BODY_JOINT_RIGHT_HAND_LITTLE_PROXIMAL_FB = 66,
XR_BODY_JOINT_RIGHT_HAND_LITTLE_INTERMEDIATE_FB = 67,
XR_BODY_JOINT_RIGHT_HAND_LITTLE_DISTAL_FB = 68,
XR_BODY_JOINT_RIGHT_HAND_LITTLE_TIP_FB = 69,
XR_BODY_JOINT_COUNT_FB = 70,
XR_BODY_JOINT_NONE_FB = -1,
XR_BODY_JOINT_MAX_ENUM_FB = 0x7FFFFFFF
} XrBodyJointFB;

```

The backward (+Z) direction is parallel to the corresponding bone and points away from the finger tip. The up (+Y) direction is pointing out of the back of and perpendicular to the corresponding finger nail at the fully opened hand pose. The X direction is perpendicular to Y and Z and follows the right hand rule.

The wrist joint is located at the pivot point of the wrist, which is location invariant when twisting the hand without moving the forearm. The backward (+Z) direction is parallel to the line from wrist joint to middle finger metacarpal joint, and points away from the finger tips. The up (+Y) direction points out towards back of the hand and perpendicular to the skin at wrist. The X direction is perpendicular to the Y and Z directions and follows the right hand rule.

The palm joint is located at the center of the middle finger's metacarpal bone. The backward (+Z) direction is parallel to the middle finger's metacarpal bone, and points away from the finger tips. The up (+Y) direction is perpendicular to palm surface and pointing towards the back of the hand. The X direction is perpendicular to the Y and Z directions and follows the right hand rule.

Body skeleton has the full set of body joints (e.g. defined by [XrBodyJointFB](#)), organized in a hierarchy with a default T-shape body pose.

The purpose of the skeleton is to provide data about the body size. Coordinates are relative to each other, so there is no any relation to any space.

The calculation of the body size **may** be updated during a session. Each time the calculation of the size is changed, `skeletonChangedCount` of [XrBodyJointLocationsFB](#) is changed to indicate that a new skeleton **may** be retrieved.

### New Object Types

- [XrBodyTrackerFB](#)

### New Flag Types

### New Enum Constants

- `XR_BODY_JOINT_COUNT_FB`

[XrObjectType](#) enumeration is extended with:

- `XR_OBJECT_TYPE_BODY_TRACKER_FB`

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SYSTEM_BODY_TRACKING_PROPERTIES_FB`
- `XR_TYPE_BODY_TRACKER_CREATE_INFO_FB`
- `XR_TYPE_BODY_JOINTS_LOCATE_INFO_FB`
- `XR_TYPE_BODY_JOINT_LOCATIONS_FB`
- `XR_TYPE_BODY_SKELETON_FB`

### New Enums

- [XrBodyJointFB](#)
- [XrBodyJointSetFB](#)

### New Structures

- [XrSystemBodyTrackingPropertiesFB](#)
- [XrBodyTrackerCreateInfoFB](#)

- [XrBodyJointsLocateInfoFB](#)
- [XrBodyJointLocationFB](#)
- [XrBodyJointLocationsFB](#)
- [XrBodySkeletonJointFB](#)
- [XrBodySkeletonFB](#)

### **New Functions**

- [xrCreateBodyTrackerFB](#)
- [xrDestroyBodyTrackerFB](#)
- [xrLocateBodyJointsFB](#)
- [xrGetBodySkeletonFB](#)

### **Issues**

### **Version History**

- Revision 1, 2022-07-18 (Igor Tceglevskii)
  - Initial extension description

## **12.44. XR\_FB\_color\_space**

### **Name String**

`XR_FB_color_space`

### **Extension Type**

Instance extension

### **Registered Extension Number**

109

### **Revision**

3

### **Extension and Version Dependencies**

[OpenXR 1.0](#)

### **Contributors**

Volga Aksoy, Facebook  
Cass Everitt, Facebook  
Gloria Kennickell, Facebook

### **Overview**

XR devices may use a color space that is different from many monitors used in development. Application developers may desire to specify the color space in which they have authored their application so appropriate colors are shown when the application is running on the XR device.

This extension allows:

- An application to get the native color space of the XR device.
- An application to enumerate the supported color spaces for the session.
- An application to set the color space for the session.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SYSTEM_COLOR_SPACE_PROPERTIES_FB`

`XrResult` enumeration is extended with:

- `XR_ERROR_COLOR_SPACE_UNSUPPORTED_FB`

## New Enums

The possible color spaces are specified by the `XrColorSpaceFB` enumeration.

```
// Provided by XR_FB_color_space
typedef enum XrColorSpaceFB {
    XR_COLOR_SPACE_UNMANAGED_FB = 0,
    XR_COLOR_SPACE_REC2020_FB = 1,
    XR_COLOR_SPACE_REC709_FB = 2,
    XR_COLOR_SPACE_RIFT_CV1_FB = 3,
    XR_COLOR_SPACE_RIFT_S_FB = 4,
    XR_COLOR_SPACE_QUEST_FB = 5,
    XR_COLOR_SPACE_P3_FB = 6,
    XR_COLOR_SPACE_ADOBE_RGB_FB = 7,
    XR_COLOR_SPACE_MAX_ENUM_FB = 0x7FFFFFFF
} XrColorSpaceFB;
```

## Enumerant Descriptions

- **XR\_COLOR\_SPACE\_UNMANAGED\_FB**. No color correction, not recommended for production use.
- **XR\_COLOR\_SPACE\_REC2020\_FB**. Standard Rec. 2020 chromacities with D65 white point.
- **XR\_COLOR\_SPACE\_REC709\_FB**. Standard Rec. 709 chromaticities, similar to sRGB.
- **XR\_COLOR\_SPACE\_RIFT\_CV1\_FB**. Unique color space, between P3 and Adobe RGB using D75 white point. This is the preferred color space for standardized color across all Oculus HMDs.

Color Space Details with Chromacity Primaries in CIE 1931 xy:

- Red: (0.666, 0.334)
  - Green: (0.238, 0.714)
  - Blue: (0.139, 0.053)
  - White: (0.298, 0.318)
- **XR\_COLOR\_SPACE\_RIFT\_S\_FB**. Unique color space. Similar to Rec 709 using D75.

Color Space Details with Chromacity Primaries in CIE 1931 xy:

- Red: (0.640, 0.330)
  - Green: (0.292, 0.586)
  - Blue: (0.156, 0.058)
  - White: (0.298, 0.318)
- **XR\_COLOR\_SPACE\_QUEST\_FB**. Unique color space. Similar to Rift CV1 using D75 white point

Color Space Details with Chromacity Primaries in CIE 1931 xy:

- Red: (0.661, 0.338)
  - Green: (0.228, 0.718)
  - Blue: (0.142, 0.042)
  - White: (0.298, 0.318)
- **XR\_COLOR\_SPACE\_P3\_FB**. Similar to DCI-P3, but uses D65 white point instead.

Color Space Details with Chromacity Primaries in CIE 1931 xy:

- Red: (0.680, 0.320)
  - Green: (0.265, 0.690)
  - Blue: (0.150, 0.060)
  - White: (0.313, 0.329)
- **XR\_COLOR\_SPACE\_ADOBE\_RGB\_FB**. Standard Adobe chromacities.

## New Structures

An application may inspect the native color space of the system by chaining an [XrSystemColorSpacePropertiesFB](#) structure to the [XrSystemProperties](#) when calling [xrGetSystemProperties](#).

The [XrSystemColorSpacePropertiesFB](#) structure is defined as:

```
// Provided by XR_FB_color_space
typedef struct XrSystemColorSpacePropertiesFB {
    XrStructureType    type;
    void*              next;
    XrColorSpaceFB     colorSpace;
} XrSystemColorSpacePropertiesFB;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `colorSpace` is the native color space of the XR device.

### Valid Usage (Implicit)

- The [XR\\_FB\\_color\\_space](#) extension **must** be enabled prior to using [XrSystemColorSpacePropertiesFB](#)
- `type` **must** be `XR_TYPE_SYSTEM_COLOR_SPACE_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrEnumerateColorSpacesFB](#) function is defined as:

```
// Provided by XR_FB_color_space
XrResult xrEnumerateColorSpacesFB(
    XrSession session,
    uint32_t colorSpaceCapacityInput,
    uint32_t* colorSpaceCountOutput,
    XrColorSpaceFB* colorSpaces);
```

## Parameter Descriptions

- `session` is the session that enumerates the supported color spaces.
- `colorSpaceCapacityInput` is the capacity of the `colorSpaces` array, or 0 to retrieve the required capacity.
- `colorSpaceCountOutput` is a pointer to the count of `XrColorSpaceFB` `colorSpaces` written, or a pointer to the required capacity in the case that `colorSpaceCapacityInput` is insufficient.
- `colorSpaces` is a pointer to an array of `XrColorSpaceFB` color spaces, but **can** be `NULL` if `colorSpaceCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `colorSpaces` size.

`xrEnumerateColorSpacesFB` enumerates the color spaces supported by the current session. Runtimes **must** always return identical buffer contents from this enumeration for the lifetime of the session.

## Valid Usage (Implicit)

- The `XR_FB_color_space` extension **must** be enabled prior to calling `xrEnumerateColorSpacesFB`
- `session` **must** be a valid `XrSession` handle
- `colorSpaceCountOutput` **must** be a pointer to a `uint32_t` value
- If `colorSpaceCapacityInput` is not 0, `colorSpaces` **must** be a pointer to an array of `colorSpaceCapacityInput` `XrColorSpaceFB` values



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `xrSetColorSpaceFB` function is defined as:

```
// Provided by XR_FB_color_space
XrResult xrSetColorSpaceFB(
    XrSession session,
    const XrColorSpaceFB colorSpace);
```

## Parameter Descriptions

- `session` is a valid `XrSession` handle.
- `colorSpace` is a supported color space. Supported color spaces are indicated by `xrEnumerateColorSpacesFB`.

`xrSetColorSpaceFB` provides a mechanism for an application to specify the color space used in the final rendered frame. If this function is not called, the session will use the color space deemed appropriate by the runtime. Oculus HMDs for both PC and Mobile product lines default to `XR_COLOR_SPACE_RIFT_CV1_FB`. The runtime **must** return `XR_ERROR_COLOR_SPACE_UNSUPPORTED_FB` if `colorSpace` is not one of the values enumerated by `xrEnumerateColorSpacesFB`.

Formal definitions of color spaces contain a number of aspects such as gamma correction, max luminance and more. However, `xrSetColorSpaceFB` will only affect the color gamut of the output by transforming the color gamut from the source (defined by the `colorSpace` parameter) to the HMD

display's color gamut (defined by the hardware internally). This call will not affect gamma correction, leaving that to follow the GPU texture format standards. Luminance, tonemapping, and other aspects of the color space will also remain unaffected.

For more info on color management in Oculus HMDs, please refer to this guide: [Color Management in Oculus Headsets](#)

### Valid Usage (Implicit)

- The `XR_FB_color_space` extension **must** be enabled prior to calling `xrSetColorSpaceFB`
- `session` **must** be a valid `XrSession` handle
- `colorSpace` **must** be a valid `XrColorSpaceFB` value

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_COLOR_SPACE_UNSUPPORTED_FB`

### Issues

### Version History

- Revision 1, 2020-11-09 (Gloria Kennickell)
  - Initial extension description
- Revision 2, 2021-09-28 (Rylie Pavlik, Collabora, Ltd.)
  - Fix XML markup to indicate that `XrSystemColorSpacePropertiesFB` is chained to `XrSystemProperties`.

- Revision 3, 2022-09-01 (Rylie Pavlik, Collabora, Ltd.)
  - Fix XML markup to indicate that `XrSystemColorSpacePropertiesFB` is returned-only.

## 12.45. XR\_FB\_composition\_layer\_alpha\_blend

### Name String

`XR_FB_composition_layer_alpha_blend`

### Extension Type

Instance extension

### Registered Extension Number

42

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Cass Everitt, Facebook  
Gloria Kennickell, Facebook  
Johannes Schmid, Facebook

### Overview

This extension provides explicit control over source and destination blend factors, with separate controls for color and alpha. When specified, these blend controls supersede the behavior of `XR_COMPOSITION_LAYER_BLEND_TEXTURE_SOURCE_ALPHA_BIT`.

When `XR_COMPOSITION_LAYER_UNPREMULTIPLIED_ALPHA_BIT` is specified, the source color is unpremultiplied alpha.

Like color, destination alpha is initialized to 0 before composition begins.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

These blend factors are specified by attaching a `XrCompositionLayerAlphaBlendFB` structure to the `next` chain of a layer structure derived from `XrCompositionLayerBaseHeader`.

### New Object Types

### New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_ALPHA\\_BLEND\\_FB](#)

## New Enums

The possible blend factors are specified by the [XrBlendFactorFB](#) enumeration.

```
// Provided by XR_FB_composition_layer_alpha_blend
typedef enum XrBlendFactorFB {
    XR_BLEND_FACTOR_ZERO_FB = 0,
    XR_BLEND_FACTOR_ONE_FB = 1,
    XR_BLEND_FACTOR_SRC_ALPHA_FB = 2,
    XR_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA_FB = 3,
    XR_BLEND_FACTOR_DST_ALPHA_FB = 4,
    XR_BLEND_FACTOR_ONE_MINUS_DST_ALPHA_FB = 5,
    XR_BLEND_FACTOR_MAX_ENUM_FB = 0x7FFFFFFF
} XrBlendFactorFB;
```

## New Structures

The [XrCompositionLayerAlphaBlendFB](#) structure is defined as:

```
// Provided by XR_FB_composition_layer_alpha_blend
typedef struct XrCompositionLayerAlphaBlendFB {
    XrStructureType    type;
    void*              next;
    XrBlendFactorFB    srcFactorColor;
    XrBlendFactorFB    dstFactorColor;
    XrBlendFactorFB    srcFactorAlpha;
    XrBlendFactorFB    dstFactorAlpha;
} XrCompositionLayerAlphaBlendFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `srcFactorColor` specifies the source color blend factor.
- `dstFactorColor` specifies the destination color blend factor.
- `srcFactorAlpha` specifies the source alpha blend factor.
- `dstFactorAlpha` specifies the destination alpha blend factor.

[XrCompositionLayerAlphaBlendFB](#) provides applications with explicit control over source and destination blend factors.

The [XrCompositionLayerAlphaBlendFB](#) structure **must** be provided in the `next` chain of the [XrCompositionLayerBaseHeader](#) structure.

## Valid Usage (Implicit)

- The [XR\\_FB\\_composition\\_layer\\_alpha\\_blend](#) extension **must** be enabled prior to using [XrCompositionLayerAlphaBlendFB](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_ALPHA_BLEND_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `srcFactorColor` **must** be a valid [XrBlendFactorFB](#) value
- `dstFactorColor` **must** be a valid [XrBlendFactorFB](#) value
- `srcFactorAlpha` **must** be a valid [XrBlendFactorFB](#) value
- `dstFactorAlpha` **must** be a valid [XrBlendFactorFB](#) value

## New Functions

## Issues

- Should we add separate blend controls for color and alpha?
  - Yes. New use cases necessitated adding separate blend controls for color and alpha.

## Version History

- Revision 1, 2020-06-22 (Gloria Kennickell)
  - Initial draft
- Revision 2, 2020-06-22 (Gloria Kennickell)

- Provide separate controls for color and alpha blend factors.

## 12.46. XR\_FB\_composition\_layer\_depth\_test

### Name String

`XR_FB_composition_layer_depth_test`

### Extension Type

Instance extension

### Registered Extension Number

213

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Guodong Rong, Meta

Cass Everitt, Meta

Jian Zhang, Meta

### Overview

This extension enables depth-tested layer composition. The compositor will maintain a depth buffer in addition to a color buffer. The depth buffer is cleared to a depth corresponding to the infinitely far distance at the beginning of composition.

When composing each layer, if depth testing is requested, the incoming layer depths are transformed into the compositor window space depth and compared to the depth stored in the frame buffer. After the transformation, incoming depths that are outside of the range of the compositor window space depth **must** be clamped. If the depth test fails, the fragment is discarded. If the depth test passes the depth buffer is updated if depth writes are enabled, and color processing continues.

Depth testing requires depth values for the layer. For projection layers, this can be supplied via the [XR\\_KHR\\_composition\\_layer\\_depth](#) extension. For geometric primitive layers, the runtime computes the depth of the sample directly from the layer parameters. An [XrCompositionLayerDepthTestFB](#) chained to layers without depth **must** be ignored.

### New Object Types

### New Flag Types

### New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_DEPTH\\_TEST\\_FB](#)

## New Enums

The possible comparison operations are specified by the [XrCompareOpFB](#) enumeration.

```
// Provided by XR_FB_composition_layer_depth_test
typedef enum XrCompareOpFB {
    XR_COMPARE_OP_NEVER_FB = 0,
    XR_COMPARE_OP_LESS_FB = 1,
    XR_COMPARE_OP_EQUAL_FB = 2,
    XR_COMPARE_OP_LESS_OR_EQUAL_FB = 3,
    XR_COMPARE_OP_GREATER_FB = 4,
    XR_COMPARE_OP_NOT_EQUAL_FB = 5,
    XR_COMPARE_OP_GREATER_OR_EQUAL_FB = 6,
    XR_COMPARE_OP_ALWAYS_FB = 7,
    XR_COMPARE_OP_MAX_ENUM_FB = 0x7FFFFFFF
} XrCompareOpFB;
```

## Enumerant Descriptions

- [XR\\_COMPARE\\_OP\\_NEVER\\_FB](#) — Comparison is never true.
- [XR\\_COMPARE\\_OP\\_LESS\\_FB](#) — Comparison is true if source less than is destination.
- [XR\\_COMPARE\\_OP\\_EQUAL\\_FB](#) — Comparison is true if source is equal to destination.
- [XR\\_COMPARE\\_OP\\_LESS\\_OR\\_EQUAL\\_FB](#) — Comparison is true if source is less than or equal to destination.
- [XR\\_COMPARE\\_OP\\_GREATER\\_FB](#) — Comparison is true if source is greater than destination.
- [XR\\_COMPARE\\_OP\\_NOT\\_EQUAL\\_FB](#) — Comparison is true if source is not equal to destination.
- [XR\\_COMPARE\\_OP\\_GREATER\\_OR\\_EQUAL\\_FB](#) — Comparison is true if source is greater than or equal to destination.
- [XR\\_COMPARE\\_OP\\_ALWAYS\\_FB](#) — Comparison is always true.

## New Structures

The [XrCompositionLayerDepthTestFB](#) structure is defined as:

```
// Provided by XR_FB_composition_layer_depth_test
typedef struct XrCompositionLayerDepthTestFB {
    XrStructureType    type;
    const void*        next;
    XrBool32           depthMask;
    XrCompareOpFB      compareOp;
} XrCompositionLayerDepthTestFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `depthMask` is a boolean indicating whether writes to the composition depth buffer are enabled.
- `compareOp` is an enum that indicates which compare operation is used in the depth test.

To specify that a layer should be depth tested, a [XrCompositionLayerDepthTestFB](#) structure **must** be passed via the polymorphic [XrCompositionLayerBaseHeader](#) structure's `next` parameter chain.

## Valid Usage (Implicit)

- The [XR\\_FB\\_composition\\_layer\\_depth\\_test](#) extension **must** be enabled prior to using [XrCompositionLayerDepthTestFB](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_DEPTH_TEST_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `compareOp` **must** be a valid [XrCompareOpFB](#) value

## New Functions

## Issues

## Version History

- Revision 1, 2022-02-17 (Cass Everitt)
  - Initial draft

# 12.47. XR\_FB\_composition\_layer\_image\_layout

## Name String

`XR_FB_composition_layer_image_layout`



## Extension Type

Instance extension

## Registered Extension Number

41

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

## Overview

This extension does not define a new composition layer type, but rather it defines parameters that change the interpretation of the image layout, where the default image layout is dictated by the Graphics API.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

```
typedef XrFlags64 XrCompositionLayerImageLayoutFlagsFB;
```

```
// Flag bits for XrCompositionLayerImageLayoutFlagsFB  
static const XrCompositionLayerImageLayoutFlagsFB  
XR_COMPOSITION_LAYER_IMAGE_LAYOUT_VERTICAL_FLIP_BIT_FB = 0x00000001;
```

## Flag Descriptions

- `XR_COMPOSITION_LAYER_IMAGE_LAYOUT_VERTICAL_FLIP_BIT_FB` indicates the coordinate origin must be considered flipped vertically.

### New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_IMAGE_LAYOUT_FB`

### New Enums

- `XR_COMPOSITION_LAYER_IMAGE_LAYOUT_VERTICAL_FLIP_BIT_FB`

### New Structures

The `XrCompositionLayerImageLayoutFB` structure is defined as:

```
// Provided by XR_FB_composition_layer_image_layout
typedef struct XrCompositionLayerImageLayoutFB {
    XrStructureType          type;
    void*                    next;
    XrCompositionLayerImageLayoutFlagsFB  flags;
} XrCompositionLayerImageLayoutFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of `XrCompositionLayerImageLayoutFlagBitsFB`.

`XrCompositionLayerImageLayoutFB` contains additional flags used to change the interpretation of the image layout for a composition layer.

To specify the additional flags, you **must** create a `XrCompositionLayerImageLayoutFB` structure and pass it via the `XrCompositionLayerBaseHeader` structure's `next` parameter.

## Valid Usage (Implicit)

- The `XR_FB_composition_layer_image_layout` extension **must** be enabled prior to using `XrCompositionLayerImageLayoutFB`
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_IMAGE_LAYOUT_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be `0` or a valid combination of `XrCompositionLayerImageLayoutFlagBitsFB` values

### New Functions

### Issues

### Version History

- Revision 1, 2020-07-06 (Gloria Kennickell)
  - Initial draft

## 12.48. XR\_FB\_composition\_layer\_secure\_content

### Name String

`XR_FB_composition_layer_secure_content`

### Extension Type

Instance extension

### Registered Extension Number

73

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

### Overview

This extension does not define a new composition layer type, but rather it provides support for the application to specify an existing composition layer type has secure content and whether it must be completely excluded from external outputs, like video or screen capture, or if proxy content must be

rendered in its place.

In order to enable the functionality of this extension, you **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

```
typedef XrFlags64 XrCompositionLayerSecureContentFlagsFB;
```

```
// Flag bits for XrCompositionLayerSecureContentFlagsFB
static const XrCompositionLayerSecureContentFlagsFB
XR_COMPOSITION_LAYER_SECURE_CONTENT_EXCLUDE_LAYER_BIT_FB = 0x00000001;
static const XrCompositionLayerSecureContentFlagsFB
XR_COMPOSITION_LAYER_SECURE_CONTENT_REPLACE_LAYER_BIT_FB = 0x00000002;
```

## Flag Descriptions

- `XR_COMPOSITION_LAYER_SECURE_CONTENT_EXCLUDE_LAYER_BIT_FB` — Indicates the layer will only be visible inside the HMD, and not visible to external sources
- `XR_COMPOSITION_LAYER_SECURE_CONTENT_REPLACE_LAYER_BIT_FB` — Indicates the layer will be displayed inside the HMD, but replaced by proxy content when written to external sources

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_SECURE_CONTENT_FB`

## New Enums

- `XR_COMPOSITION_LAYER_SECURE_CONTENT_EXCLUDE_LAYER_BIT_FB`
- `XR_COMPOSITION_LAYER_SECURE_CONTENT_REPLACE_LAYER_BIT_FB`

## New Structures

The `XrCompositionLayerSecureContentFB` structure is defined as:

```
// Provided by XR_FB_composition_layer_secure_content
typedef struct XrCompositionLayerSecureContentFB {
    XrStructureType          type;
    const void*              next;
    XrCompositionLayerSecureContentFlagsFB flags;
} XrCompositionLayerSecureContentFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of [XrCompositionLayerSecureContentFlagBitsFB](#).

[XrCompositionLayerSecureContentFB](#) contains additional flags to indicate a composition layer contains secure content and must not be written to external outputs.

If both [XR\\_COMPOSITION\\_LAYER\\_SECURE\\_CONTENT\\_EXCLUDE\\_LAYER\\_BIT\\_FB](#) and [XR\\_COMPOSITION\\_LAYER\\_SECURE\\_CONTENT\\_REPLACE\\_LAYER\\_BIT\\_FB](#) are set, [XR\\_COMPOSITION\\_LAYER\\_SECURE\\_CONTENT\\_EXCLUDE\\_LAYER\\_BIT\\_FB](#) will take precedence.

To specify the additional flags, you **must** create a [XrCompositionLayerSecureContentFB](#) structure and pass it via the [XrCompositionLayerBaseHeader](#) structure's `next` parameter.

## Valid Usage (Implicit)

- The [XR\\_FB\\_composition\\_layer\\_secure\\_content](#) extension **must** be enabled prior to using [XrCompositionLayerSecureContentFB](#)
- `type` **must** be [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_SECURE\\_CONTENT\\_FB](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be a valid combination of [XrCompositionLayerSecureContentFlagBitsFB](#) values
- `flags` **must** not be `0`

## New Functions

## Issues

## Version History

- Revision 1, 2020-06-16 (Gloria Kennickell)
  - Initial draft

## 12.49. XR\_FB\_composition\_layer\_settings

### Name String

`XR_FB_composition_layer_settings`

### Extension Type

Instance extension

### Registered Extension Number

205

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Grant Yang, Meta Platforms

### Overview

This extension allows applications to request the use of processing options such as sharpening or super-sampling on a composition layer.

In order to enable the functionality of this extension, you **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

### New Object Types

### New Flag Types

```
typedef XrFlags64 XrCompositionLayerSettingsFlagsFB;
```

```
// Flag bits for XrCompositionLayerSettingsFlagsFB
static const XrCompositionLayerSettingsFlagsFB
XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SUPER_SAMPLING_BIT_FB = 0x00000001;
static const XrCompositionLayerSettingsFlagsFB
XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SUPER_SAMPLING_BIT_FB = 0x00000002;
static const XrCompositionLayerSettingsFlagsFB
XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SHARPENING_BIT_FB = 0x00000004;
static const XrCompositionLayerSettingsFlagsFB
XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SHARPENING_BIT_FB = 0x00000008;
static const XrCompositionLayerSettingsFlagsFB
XR_COMPOSITION_LAYER_SETTINGS_AUTO_LAYER_FILTER_BIT_META = 0x00000020;
```

## Flag Descriptions

- `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SUPER_SAMPLING_BIT_FB` — Indicates compositor **may** use layer texture supersampling.
- `XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SUPER_SAMPLING_BIT_FB` — Indicates compositor **may** use high quality layer texture supersampling.
- `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SHARPENING_BIT_FB` — Indicates compositor **may** use layer texture sharpening.
- `XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SHARPENING_BIT_FB` — Indicates compositor **may** use high quality layer texture sharpening.
- `XR_COMPOSITION_LAYER_SETTINGS_AUTO_LAYER_FILTER_BIT_META` — Indicates compositor **may** automatically toggle a texture filtering mechanism to improve visual quality of layer. This **must** not be the only bit set. (Added by `XR_META_automatic_layer_filter`) (Added by the `XR_META_automatic_layer_filter` extension)

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_SETTINGS_FB`

## New Enums

- `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SUPER_SAMPLING_BIT_FB`
- `XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SUPER_SAMPLING_BIT_FB`
- `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SHARPENING_BIT_FB`
- `XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SHARPENING_BIT_FB`

## New Structures

The [XrCompositionLayerSettingsFB](#) structure is defined as:

```
// Provided by XR_FB_composition_layer_settings
typedef struct XrCompositionLayerSettingsFB {
    XrStructureType          type;
    const void*             next;
    XrCompositionLayerSettingsFlagsFB layerFlags;
} XrCompositionLayerSettingsFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layerFlags` is a bitmask of [XrCompositionLayerSettingsFlagBitsFB](#).

[XrCompositionLayerSettingsFB](#) contains additional flags to indicate which processing steps to perform on a composition layer.

If both `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SUPER_SAMPLING_BIT_FB` and `XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SUPER_SAMPLING_BIT_FB` are set, `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SUPER_SAMPLING_BIT_FB` will take precedence.

If both `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SHARPENING_BIT_FB` and `XR_COMPOSITION_LAYER_SETTINGS_QUALITY_SHARPENING_BIT_FB` are set, `XR_COMPOSITION_LAYER_SETTINGS_NORMAL_SHARPENING_BIT_FB` will take precedence.

To specify the additional flags, create an [XrCompositionLayerSettingsFB](#) structure and pass it via the [XrCompositionLayerBaseHeader](#) structure's `next` parameter.

## Valid Usage (Implicit)

- The `XR_FB_composition_layer_settings` extension **must** be enabled prior to using [XrCompositionLayerSettingsFB](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_SETTINGS_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layerFlags` **must** be a valid combination of [XrCompositionLayerSettingsFlagBitsFB](#) values
- `layerFlags` **must** not be `0`



## New Functions

## Issues

## Version History

- Revision 1, 2022-03-08 (Grant Yang)
  - Initial draft

# 12.50. XR\_FB\_display\_refresh\_rate

## Name String

`XR_FB_display_refresh_rate`

## Extension Type

Instance extension

## Registered Extension Number

102

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## IP Status

No known IP claims.

## Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

## Overview

On platforms which support dynamically adjusting the display refresh rate, application developers may request a specific display refresh rate in order to improve the overall user experience, examples include:

- A video application may choose a display refresh rate which better matches the video content playback rate in order to achieve smoother video frames.
- An application which can support a higher frame rate may choose to render at the higher rate to improve the overall perceptual quality, for example, lower latency and less flicker.

This extension allows:

- An application to identify what display refresh rates the session supports and the current display refresh rate.
- An application to request a display refresh rate to indicate its preference to the runtime.
- An application to receive notification of changes to the display refresh rate which are delivered via events.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_EVENT_DATA_DISPLAY_REFRESH_RATE_CHANGED_FB`

[XrResult](#) enumeration is extended with:

- `XR_ERROR_DISPLAY_REFRESH_RATE_UNSUPPORTED_FB`

## New Enums

## New Structures

Receiving the [XrEventDataDisplayRefreshRateChangedFB](#) event structure indicates that the display refresh rate has changed.

The [XrEventDataDisplayRefreshRateChangedFB](#) structure is defined as:

```
// Provided by XR_FB_display_refresh_rate
typedef struct XrEventDataDisplayRefreshRateChangedFB {
    XrStructureType    type;
    const void*        next;
    float               fromDisplayRefreshRate;
    float               toDisplayRefreshRate;
} XrEventDataDisplayRefreshRateChangedFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `fromDisplayRefreshRate` is the previous display refresh rate.
- `toDisplayRefreshRate` is the new display refresh rate.

## Valid Usage (Implicit)

- The [XR\\_FB\\_display\\_refresh\\_rate](#) extension **must** be enabled prior to using [XrEventDataDisplayRefreshRateChangedFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_DISPLAY_REFRESH_RATE_CHANGED_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrEnumerateDisplayRefreshRatesFB](#) function is defined as:

```
// Provided by XR_FB_display_refresh_rate
XrResult xrEnumerateDisplayRefreshRatesFB(
    XrSession                session,
    uint32_t                 displayRefreshRateCapacityInput,
    uint32_t*                displayRefreshRateCountOutput,
    float*                   displayRefreshRates);
```

## Parameter Descriptions

- `session` is the session that enumerates the supported display refresh rates.
- `displayRefreshRateCapacityInput` is the capacity of the `displayRefreshRates`, or 0 to retrieve the required capacity.
- `displayRefreshRateCountOutput` is a pointer to the count of `float displayRefreshRates` written, or a pointer to the required capacity in the case that `displayRefreshRateCapacityInput` is insufficient.
- `displayRefreshRates` is a pointer to an array of `float` display refresh rates, but **can** be `NULL` if `displayRefreshRateCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `displayRefreshRates` size.

`xrEnumerateDisplayRefreshRatesFB` enumerates the display refresh rates supported by the current session. Display refresh rates **must** be in order from lowest to highest supported display refresh rates. Runtimes **must** always return identical buffer contents from this enumeration for the lifetime of the session.

## Valid Usage (Implicit)

- The `XR_FB_display_refresh_rate` extension **must** be enabled prior to calling `xrEnumerateDisplayRefreshRatesFB`
- `session` **must** be a valid `XrSession` handle
- `displayRefreshRateCountOutput` **must** be a pointer to a `uint32_t` value
- If `displayRefreshRateCapacityInput` is not 0, `displayRefreshRates` **must** be a pointer to an array of `displayRefreshRateCapacityInput` `float` values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `xrGetDisplayRefreshRateFB` function is defined as:

```
// Provided by XR_FB_display_refresh_rate
XrResult xrGetDisplayRefreshRateFB(
    XrSession          session,
    float*             displayRefreshRate);
```

## Parameter Descriptions

- `session` is the `XrSession` to query.
- `displayRefreshRate` is a pointer to a float into which the current display refresh rate will be placed.

`xrGetDisplayRefreshRateFB` retrieves the current display refresh rate.

## Valid Usage (Implicit)

- The `XR_FB_display_refresh_rate` extension **must** be enabled prior to calling `xrGetDisplayRefreshRateFB`
- `session` **must** be a valid `XrSession` handle
- `displayRefreshRate` **must** be a pointer to a `float` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrRequestDisplayRefreshRateFB` function is defined as:

```
// Provided by XR_FB_display_refresh_rate
XrResult xrRequestDisplayRefreshRateFB(
    XrSession          session,
    float              displayRefreshRate);
```

## Parameter Descriptions

- `session` is a valid `XrSession` handle.
- `displayRefreshRate` is `0.0f` or a supported display refresh rate. Supported display refresh rates are indicated by `xrEnumerateDisplayRefreshRatesFB`.

`xrRequestDisplayRefreshRateFB` provides a mechanism for an application to request the system to

dynamically change the display refresh rate to the application preferred value. The runtime **must** return `XR_ERROR_DISPLAY_REFRESH_RATE_UNSUPPORTED_FB` if `displayRefreshRate` is not either `0.0f` or one of the values enumerated by `xrEnumerateDisplayRefreshRatesFB`. A display refresh rate of `0.0f` indicates the application has no preference.

Note that this is only a request and does not guarantee the system will switch to the requested display refresh rate.

### Valid Usage (Implicit)

- The `XR_FB_display_refresh_rate` extension **must** be enabled prior to calling `xrRequestDisplayRefreshRateFB`
- `session` **must** be a valid `XrSession` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_DISPLAY_REFRESH_RATE_UNSUPPORTED_FB`

### Issues

Changing the display refresh rate from its system default does not come without trade-offs. Increasing the display refresh rate puts more load on the entire system and can lead to thermal degradation. Conversely, lowering the display refresh rate can provide better thermal sustainability but at the cost of more perceptual issues, like higher latency and flickering.

### Version History

- Revision 1, 2020-10-05 (Gloria Kennickell)

- Initial extension description

## 12.51. XR\_FB\_eye\_tracking\_social

### Name String

`XR_FB_eye_tracking_social`

### Extension Type

Instance extension

### Registered Extension Number

203

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-07-17

### IP Status

No known IP claims.

### Contributors

Scott Ramsby, Meta  
Dikpal Reddy, Meta  
Igor Tceglevskii, Meta

### 12.51.1. Overview

This extension enables applications to obtain position and orientation of the user's eyes. It enables applications to render eyes in XR experiences.

This extension is intended to drive animation of avatar eyes. So, for that purpose, the runtimes **may** filter the poses in ways that are suitable for avatar eye interaction but detrimental to other use cases. This extension **should** not be used for other eye tracking purposes. For interaction, [XR\\_EXT\\_eye\\_gaze\\_interaction](#) **should** be used.

Eye tracking data is sensitive personal information and is closely linked to personal privacy and integrity. It is strongly recommended that applications that store or transfer eye tracking data always ask the user for active and specific acceptance to do so.

If a runtime supports a permission system to control application access to the eye tracker, then the



runtime **must** set the `isValid` field to `XR_FALSE` on the supplied `XrEyeGazeFB` structure until the application has been allowed access to the eye tracker. When the application access has been allowed, the runtime **may** set `isValid` on the supplied `XrEyeGazeFB` structure to `XR_TRUE`.

## 12.51.2. Inspect system capability

The `XrSystemEyeTrackingPropertiesFB` structure is defined as:

```
// Provided by XR_FB_eye_tracking_social
typedef struct XrSystemEyeTrackingPropertiesFB {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsEyeTracking;
} XrSystemEyeTrackingPropertiesFB;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsEyeTracking` is an `XrBool32`, indicating if the current system is capable of receiving eye tracking input.

An application **can** inspect whether the system is capable of eye tracking input by extending the `XrSystemProperties` with `XrSystemEyeTrackingPropertiesFB` structure when calling `xrGetSystemProperties`.

If a runtime returns `XR_FALSE` for `supportsEyeTracking`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateEyeTrackerFB`.

### Valid Usage (Implicit)

- The `XR_FB_eye_tracking_social` extension **must** be enabled prior to using `XrSystemEyeTrackingPropertiesFB`
- `type` **must** be `XR_TYPE_SYSTEM_EYE_TRACKING_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.51.3. Create an eye tracker handle

The [XrEyeTrackerFB](#) handle represents the resources for eye tracking.

```
// Provided by XR_FB_eye_tracking_social
XR_DEFINE_HANDLE(XrEyeTrackerFB)
```

This handle is used for getting eye gaze using [xrGetEyeGazesFB](#) function.

An eye tracker provides eye gaze directions.

An application creates an [XrEyeTrackerFB](#) handle using [xrCreateEyeTrackerFB](#) function.

```
// Provided by XR_FB_eye_tracking_social
XrResult xrCreateEyeTrackerFB(
    XrSession session,
    const XrEyeTrackerCreateInfoFB* createInfo,
    XrEyeTrackerFB* eyeTracker);
```

#### Parameter Descriptions

- **session** is an [XrSession](#) in which the eye tracker will be active.
- **createInfo** is the [XrEyeTrackerCreateInfoFB](#) used to specify the eye tracker.
- **eyeTracker** is the returned [XrEyeTrackerFB](#) handle.

If the system does not support eye tracking, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateEyeTrackerFB](#). In this case, the runtime **must** return `XR_FALSE` for [XrSystemEyeTrackingPropertiesFB::supportsEyeTracking](#) when the function [xrGetSystemProperties](#) is called, so that the application **can** avoid creating an eye tracker.

## Valid Usage (Implicit)

- The `XR_FB_eye_tracking_social` extension **must** be enabled prior to calling `xrCreateEyeTrackerFB`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrEyeTrackerCreateInfoFB` structure
- `eyeTracker` **must** be a pointer to an `XrEyeTrackerFB` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrEyeTrackerCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_eye_tracking_social
typedef struct XrEyeTrackerCreateInfoFB {
    XrStructureType    type;
    const void*        next;
} XrEyeTrackerCreateInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

The [XrEyeTrackerCreateInfoFB](#) structure describes the information to create an [XrEyeTrackerFB](#) handle.

## Valid Usage (Implicit)

- The [XR\\_FB\\_eye\\_tracking\\_social](#) extension **must** be enabled prior to using [XrEyeTrackerCreateInfoFB](#)
- `type` **must** be `XR_TYPE_EYE_TRACKER_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.51.4. Destroy an eye tracker handle

[xrDestroyEyeTrackerFB](#) function releases the `eyeTracker` and the underlying resources when the eye tracking experience is over.

```
// Provided by XR_FB_eye_tracking_social
XrResult xrDestroyEyeTrackerFB(
    XrEyeTrackerFB          eyeTracker);
```

## Parameter Descriptions

- `eyeTracker` is an [XrEyeTrackerFB](#) previously created by [xrCreateEyeTrackerFB](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_eye\\_tracking\\_social](#) extension **must** be enabled prior to calling [xrDestroyEyeTrackerFB](#)
- `eyeTracker` **must** be a valid [XrEyeTrackerFB](#) handle

## Thread Safety

- Access to `eyeTracker`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## 12.51.5. Get eye gaze

The `xrGetEyeGazesFB` function is defined as:

```
// Provided by XR_FB_eye_tracking_social
XrResult xrGetEyeGazesFB(
    XrEyeTrackerFB          eyeTracker,
    const XrEyeGazesInfoFB* gazeInfo,
    XrEyeGazesFB*          eyeGazes);
```

## Parameter Descriptions

- `eyeTracker` is an `XrEyeTrackerFB` previously created by `xrCreateEyeTrackerFB`.
- `gazeInfo` is the information to get eye gaze.
- `eyeGazes` is a pointer to `XrEyeGazesFB` receiving the returned eye poses and confidence.

The `xrGetEyeGazesFB` function obtains pose for a user's eyes at a specific time and within a specific coordinate system.

## Valid Usage (Implicit)

- The `XR_FB_eye_tracking_social` extension **must** be enabled prior to calling `xrGetEyeGazesFB`
- `eyeTracker` **must** be a valid `XrEyeTrackerFB` handle
- `gazeInfo` **must** be a pointer to a valid `XrEyeGazesInfoFB` structure
- `eyeGazes` **must** be a pointer to an `XrEyeGazesFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrEyeGazesInfoFB` structure describes the information to get eye gaze directions.

```
// Provided by XR_FB_eye_tracking_social
typedef struct XrEyeGazesInfoFB {
    XrStructureType    type;
    const void*        next;
    XrSpace             baseSpace;
    XrTime             time;
} XrEyeGazesInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `baseSpace` is an [XrSpace](#) within which the returned eye poses will be represented.
- `time` is an [XrTime](#) at which the eye gaze information is requested.

The application **should** request a time equal to the predicted display time for the rendered frame. The system will employ appropriate modeling to provide eye gaze at this time.

## Valid Usage (Implicit)

- The [XR\\_FB\\_eye\\_tracking\\_social](#) extension **must** be enabled prior to using [XrEyeGazesInfoFB](#)
- `type` **must** be `XR_TYPE_EYE_GAZES_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `baseSpace` **must** be a valid [XrSpace](#) handle

[XrEyeGazesFB](#) structure returns the state of the eye gaze directions.

```
// Provided by XR_FB_eye_tracking_social
typedef struct XrEyeGazesFB {
    XrStructureType    type;
    void*              next;
    XrEyeGazeFB        gaze[XR_EYE_POSITION_COUNT_FB];
    XrTime              time;
} XrEyeGazesFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `gaze` is an array of `XrEyeGazeFB` receiving the returned eye gaze directions.
- `time` is an `XrTime` time at which the returned eye gaze is tracked or extrapolated to. Equals the time for which the eye gaze was requested if the interpolation at the time was successful.

## Valid Usage (Implicit)

- The `XR_FB_eye_tracking_social` extension **must** be enabled prior to using `XrEyeGazesFB`
- `type` **must** be `XR_TYPE_EYE_GAZES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- Any given element of `gaze` **must** be a valid `XrEyeGazeFB` structure

`XrEyeGazeFB` structure describes the validity, direction, and confidence of a social eye gaze observation.

```
// Provided by XR_FB_eye_tracking_social
typedef struct XrEyeGazeFB {
    XrBool32    isValid;
    XrPosef     gazePose;
    float       gazeConfidence;
} XrEyeGazeFB;
```

## Member Descriptions

- `isValid` is an `XrBool32` indicating if the returned `gazePose` is valid. Callers **should** check the validity of pose prior to use.
- `gazePose` is an `XrPosef` describing the position and orientation of the user's eye. The pose is represented in the coordinate system provided by `XrEyeGazesInfoFB::baseSpace`.
- `gazeConfidence` is a `float` value between 0 and 1 that represents the confidence for eye pose. A value of 0 represents no confidence in the pose returned, and a value of 1 means maximum confidence in the returned eye pose.



If the returned `isValid` is true, the runtime **must** return `gazePose` and `gazeConfidence`.

If the returned `isValid` is false, it indicates either the eye tracker did not detect the eye gaze or the application lost input focus.

The eye gaze pose is natively oriented with +Y up, +X to the right, and -Z forward and not gravity-aligned, similar to the `XR_REFERENCE_SPACE_TYPE_VIEW`.

### Valid Usage (Implicit)

- The `XR_FB_eye_tracking_social` extension **must** be enabled prior to using `XrEyeGazeFB`

The `XrEyePositionFB` describes which eye in the specific position of the `gaze` is in the `XrEyeGazesFB`.

```
// Provided by XR_FB_eye_tracking_social
typedef enum XrEyePositionFB {
    XR_EYE_POSITION_LEFT_FB = 0,
    XR_EYE_POSITION_RIGHT_FB = 1,
    XR_EYE_POSITION_COUNT_FB = 2,
    XR_EYE_POSITION_MAX_ENUM_FB = 0x7FFFFFFF
} XrEyePositionFB;
```

### Enumerant Descriptions

- `XR_EYE_POSITION_LEFT_FB` — Specifies the position of the left eye.
- `XR_EYE_POSITION_RIGHT_FB` — Specifies the position of the right eye.

## 12.51.6. Example code for locating eye gaze

The following example code demonstrates how to locate eye gaze relative to a world space.

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized
XrSpace worldSpace; // previously initialized, e.g. from
                    // XR_REFERENCE_SPACE_TYPE_LOCAL

XrSystemEyeTrackingPropertiesFB eyeTrackingSystemProperties{
    XR_TYPE_SYSTEM_EYE_TRACKING_PROPERTIES_FB};
XrSystemProperties systemProperties{
    XR_TYPE_SYSTEM_PROPERTIES, &eyeTrackingSystemProperties};
```

```

CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
if (!eyeTrackingSystemProperties.supportsEyeTracking) {
    // The system does not support eye tracking.
    return;
}

// Get function pointer for xrCreateEyeTrackerFB.
PFN_xrCreateEyeTrackerFB pfnCreateEyeTrackerFB;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateEyeTrackerFB",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnCreateEyeTrackerFB)));

// Create an eye tracker.
XrEyeTrackerFB eyeTracker{};
{
    XrEyeTrackerCreateInfoFB createInfo{XR_TYPE_EYE_TRACKER_CREATE_INFO_FB};
    CHK_XR(pfnCreateEyeTrackerFB(session, &createInfo, &eyeTracker));
}

// Allocate buffers to receive eyes pose and confidence data before frame
// the loop starts.
XrEyeGazesFB eyeGazes{XR_TYPE_EYE_GAZES_FB};
eyeGazes.next = nullptr;

// Get function pointer for xrGetEyeGazesFB.
PFN_xrGetEyeGazesFB pfnGetEyeGazesFB;
CHK_XR(xrGetInstanceProcAddr(instance, "xrGetEyeGazesFB",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnGetEyeGazesFB)));

while (1) {
    // ...
    // For every frame in frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrEyeGazesInfoFB gazesInfo{XR_TYPE_EYE_GAZES_INFO_FB};
    gazesInfo.baseSpace = worldSpace;
    gazesInfo.time = time;

    CHK_XR(pfnGetEyeGazesFB(eyeTracker, &gazesInfo, &eyeGazes));

    if (eyeGazes.gaze[XR_EYE_POSITION_LEFT_FB].isValid) {
        // ....
    }
}

```

## New Object Types

- [XrEyeTrackerFB](#)

## New Flag Types

## New Enum Constants

[XrObjectType](#) enumeration is extended with:

- `XR_OBJECT_TYPE_EYE_TRACKER_FB`

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SYSTEM_EYE_TRACKING_PROPERTIES_FB`
- `XR_TYPE_EYE_TRACKER_CREATE_INFO_FB`
- `XR_TYPE_EYE_GAZES_INFO_FB`
- `XR_TYPE_EYE_GAZES_FB`

## New Enums

- [XrEyePositionFB](#)

## New Structures

- [XrSystemEyeTrackingPropertiesFB](#)
- [XrEyeTrackerCreateInfoFB](#)
- [XrEyeGazesInfoFB](#)
- [XrEyeGazeFB](#)
- [XrEyeGazesFB](#)

## New Functions

- [xrCreateEyeTrackerFB](#)
- [xrDestroyEyeTrackerFB](#)
- [xrGetEyeGazesFB](#)

## Issues

## Version History

- Revision 1, 2022-07-17 (Igor Tceglevskii)
  - Initial extension description

# 12.52. XR\_FB\_face\_tracking

## Name String

XR\_FB\_face\_tracking

## Extension Type

Instance extension

## Registered Extension Number

202

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-07-15

## IP Status

No known IP claims.

## Contributors

Jaebong Lee, Meta

Dikpal Reddy, Meta

Igor Tceglevskii, Meta

## 12.52.1. Overview

This extension enables applications to get weights of blend shapes. It also enables applications to render facial expressions in XR experiences.

Face tracking data is sensitive personal information and is closely linked to personal privacy and integrity. It is strongly recommended that applications storing or transferring face tracking data always ask the user for active and specific acceptance to do so.

If a runtime supports a permission system to control application access to the face tracker, then the runtime **must** set the `isValid` field to `XR_FALSE` on the supplied `XrFaceExpressionStatusFB` structure until the user allows the application to access the face tracker. When the application access has been allowed, the runtime **may** set `isValid` on the supplied `XrFaceExpressionStatusFB` structure to `XR_TRUE`.

Some permission systems **may** control access to the eye tracking separately from access to the face tracking, even though the eyes are part of the face. In case the user denied tracking of the eyes, yet, allowed tracking of the face, then the runtime **must** set the `isEyeFollowingBlendshapesValid` field to

`XR_FALSE` on the supplied `XrFaceExpressionStatusFB` for indicating that eye tracking data is not available, but at the same time **may** set the `isValid` field to `XR_TRUE` on the supplied `XrFaceExpressionStatusFB` for indicating that another part of the face is tracked properly.

### 12.52.2. Inspect system capability

```
// Provided by XR_FB_face_tracking
typedef struct XrSystemFaceTrackingPropertiesFB {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsFaceTracking;
} XrSystemFaceTrackingPropertiesFB;
```

#### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsFaceTracking` is an `XrBool32`, indicating if current system is capable of receiving face tracking input.

An application **can** inspect whether the system is capable of receiving face tracking input by extending the `XrSystemProperties` with `XrSystemFaceTrackingPropertiesFB` structure when calling `xrGetSystemProperties`.

If a runtime returns `XR_FALSE` for `supportsFaceTracking`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateFaceTrackerFB`.

#### Valid Usage (Implicit)

- The `XR_FB_face_tracking` extension **must** be enabled prior to using `XrSystemFaceTrackingPropertiesFB`
- `type` **must** be `XR_TYPE_SYSTEM_FACE_TRACKING_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.52.3. Create a face tracker handle

The `XrFaceTrackerFB` handle represents the resources for face tracking.

```
// Provided by XR_FB_face_tracking
XR_DEFINE_HANDLE(XrFaceTrackerFB)
```

This handle is used to obtain blend shapes using the [xrGetFaceExpressionWeightsFB](#) function.

The [xrCreateFaceTrackerFB](#) function is defined as:

```
// Provided by XR_FB_face_tracking
XrResult xrCreateFaceTrackerFB(
    XrSession session,
    const XrFaceTrackerCreateInfoFB* createInfo,
    XrFaceTrackerFB* faceTracker);
```

### Parameter Descriptions

- **session** is an [XrSession](#) in which the face tracker will be active.
- **createInfo** is the [XrFaceTrackerCreateInfoFB](#) used to specify the face tracker.
- **faceTracker** is the returned [XrFaceTrackerFB](#) handle.

An application **can** create an [XrFaceTrackerFB](#) handle using [xrCreateFaceTrackerFB](#) function.

If the system does not support face tracking, the runtime **must** return [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#) from [xrCreateFaceTrackerFB](#). In this case, the runtime **must** return [XR\\_FALSE](#) for [XrSystemFaceTrackingPropertiesFB::supportsFaceTracking](#) when the function [xrGetSystemProperties](#) is called, so that the application **can** avoid creating a face tracker.

### Valid Usage (Implicit)

- The [XR\\_FB\\_face\\_tracking](#) extension **must** be enabled prior to calling [xrCreateFaceTrackerFB](#)
- **session** **must** be a valid [XrSession](#) handle
- **createInfo** **must** be a pointer to a valid [XrFaceTrackerCreateInfoFB](#) structure
- **faceTracker** **must** be a pointer to an [XrFaceTrackerFB](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrFaceTrackerCreateInfoFB` structure is described as follows:

```
// Provided by XR_FB_face_tracking
typedef struct XrFaceTrackerCreateInfoFB {
    XrStructureType      type;
    const void*          next;
    XrFaceExpressionSetFB faceExpressionSet;
} XrFaceTrackerCreateInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `faceExpressionSet` is an `XrFaceExpressionSetFB` that describe the set of blend shapes to retrieve.

The `XrFaceTrackerCreateInfoFB` structure describes the information to create an `XrFaceTrackerFB` handle.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking` extension **must** be enabled prior to using `XrFaceTrackerCreateInfoFB`
- `type` **must** be `XR_TYPE_FACE_TRACKER_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `faceExpressionSet` **must** be a valid `XrFaceExpressionSetFB` value

The `XrFaceExpressionSetFB` enum describes the set of blend shapes of a facial expression to track when creating an `XrFaceTrackerFB`.

```
// Provided by XR_FB_face_tracking
typedef enum XrFaceExpressionSetFB {
    XR_FACE_EXPRESSION_SET_DEFAULT_FB = 0,
    XR_FACE_EXPRESSION_SET_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceExpressionSetFB;
```

## Enumerant Descriptions

- `XR_FACE_EXPRESSION_SET_DEFAULT_FB` — indicates that the created `XrFaceTrackerFB` tracks the set of blend shapes described by `XrFaceExpressionFB` enum, i.e. the `xrGetFaceExpressionWeightsFB` function returns an array of blend shapes with the count of `XR_FACE_EXPRESSION_COUNT_FB` and **can** be indexed using `XrFaceExpressionFB`.

```
// Provided by XR_FB_face_tracking
#define XR_FACE_EXPRESSION_SET_DEFAULT_FB XR_FACE_EXPRESSION_SET_DEFAULT_FB
```

The `XR_FACE_EXPRESSION_SET_DEFAULT_FB` is an alias for `XR_FACE_EXPRESSION_SET_DEFAULT_FB` for backward compatibility, deprecated and **should** not be used.

### 12.52.4. Delete a face tracker handle

The `xrDestroyFaceTrackerFB` function releases the `faceTracker` and the underlying resources when face tracking experience is over.



```
// Provided by XR_FB_face_tracking
XrResult xrDestroyFaceTrackerFB(
    XrFaceTrackerFB          faceTracker);
```

## Parameter Descriptions

- `faceTracker` is an `XrFaceTrackerFB` previously created by `xrCreateFaceTrackerFB`.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking` extension **must** be enabled prior to calling `xrDestroyFaceTrackerFB`
- `faceTracker` **must** be a valid `XrFaceTrackerFB` handle

## Thread Safety

- Access to `faceTracker`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## 12.52.5. Obtain facial expressions

The `xrGetFaceExpressionWeightsFB` function return blend shapes of facial expression at a given time.

```
// Provided by XR_FB_face_tracking
XrResult xrGetFaceExpressionWeightsFB(
    XrFaceTrackerFB          faceTracker,
    const XrFaceExpressionInfoFB* expressionInfo,
    XrFaceExpressionWeightsFB* expressionWeights);
```

## Parameter Descriptions

- `faceTracker` is an `XrFaceTrackerFB` previously created by `xrCreateFaceTrackerFB`.
- `expressionInfo` is a pointer to `XrFaceExpressionInfoFB` describing information to obtain face expression.
- `expressionWeights` is a pointer to `XrFaceExpressionWeightsFB` receiving the returned facial expression weights.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking` extension **must** be enabled prior to calling `xrGetFaceExpressionWeightsFB`
- `faceTracker` **must** be a valid `XrFaceTrackerFB` handle
- `expressionInfo` **must** be a pointer to a valid `XrFaceExpressionInfoFB` structure
- `expressionWeights` **must** be a pointer to an `XrFaceExpressionWeightsFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrFaceExpressionInfoFB` structure describes the information to obtain facial expression.

```
// Provided by XR_FB_face_tracking
typedef struct XrFaceExpressionInfoFB {
    XrStructureType    type;
    const void*        next;
    XrTime              time;
} XrFaceExpressionInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `time` is an [XrTime](#) at which the facial expression weights are requested.

Callers **should** request a time equal to the predicted display time for the rendered frame. The system will employ appropriate modeling to provide expressions for this time.

## Valid Usage (Implicit)

- The [XR\\_FB\\_face\\_tracking](#) extension **must** be enabled prior to using [XrFaceExpressionInfoFB](#)
- `type` **must** be `XR_TYPE_FACE_EXPRESSION_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

[XrFaceExpressionWeightsFB](#) structure returns the facial expression.

```
// Provided by XR_FB_face_tracking
typedef struct XrFaceExpressionWeightsFB {
    XrStructureType    type;
    void*              next;
    uint32_t            weightCount;
    float*              weights;
    uint32_t            confidenceCount;
    float*              confidences;
    XrFaceExpressionStatusFB status;
    XrTime              time;
} XrFaceExpressionWeightsFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `weightCount` is a `uint32_t` describing the count of elements in `weights` array.
- `weights` is a pointer to an application-allocated array of `float` that will be filled with weights of facial expression blend shapes.
- `confidenceCount` is a `uint32_t` describing the count of elements in `confidences` array.
- `confidences` is a pointer to an application-allocated array of `float` that will be filled with confidence of tracking specific parts of a face.
- `status` is the [XrFaceExpressionStatusFB](#) of validity status of the expression weights.
- `time` is an [XrTime](#) time at which the returned expression weights are tracked or extrapolated to. Equals the time at which the expression weights were requested if the extrapolating at the time was successful.

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `weightCount` is not equal to the number of blend shapes defined by the [XrFaceExpressionSetFB](#) used to create the [XrFaceTrackerFB](#).

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `confidenceCount` is not equal to the number of confidence areas defined by the [XrFaceExpressionSetFB](#) used to create the [XrFaceTrackerFB](#).

The runtime **must** return `weights` representing the weights of blend shapes of current facial expression.

The runtime **must** update the `weights` array ordered so that the application **can** index elements using the corresponding facial expression enum (e.g. [XrFaceExpressionFB](#)) as described by [XrFaceExpressionSetFB](#) when creating the [XrFaceTrackerFB](#). For example, when the [XrFaceTrackerFB](#) is created with `XR_FACE_EXPRESSION_SET_DEFAULT_FB`, the application sets the `weightCount` to `XR_FACE_EXPRESSION_COUNT_FB`, and the runtime **must** fill the `weights` array ordered so that it **can** be indexed by the [XrFaceExpressionFB](#) enum.

The runtime **must** update the `confidences` array ordered so that the application **can** index elements using the corresponding confidence area enum (e.g. [XrFaceConfidenceFB](#)) as described by [XrFaceExpressionSetFB](#) when creating the [XrFaceTrackerFB](#). For example, when the [XrFaceTrackerFB](#) is created with `XR_FACE_EXPRESSION_SET_DEFAULT_FB`, the application sets the `confidenceCount` to `XR_FACE_CONFIDENCE_COUNT_FB`, and the runtime **must** fill the `confidences` array ordered so that it **can** be indexed by the [XrFaceConfidenceFB](#) enum.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking` extension **must** be enabled prior to using `XrFaceExpressionWeightsFB`
- `type` **must** be `XR_TYPE_FACE_EXPRESSION_WEIGHTS_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `weights` **must** be a pointer to an array of `weightCount` float values
- `confidences` **must** be a pointer to an array of `confidenceCount` float values
- `status` **must** be a valid `XrFaceExpressionStatusFB` structure
- The `weightCount` parameter **must** be greater than 0
- The `confidenceCount` parameter **must** be greater than 0

`XrFaceExpressionStatusFB` structure describes the validity of facial expression weights.

```
// Provided by XR_FB_face_tracking
typedef struct XrFaceExpressionStatusFB {
    XrBool32    isValid;
    XrBool32    isEyeFollowingBlendshapesValid;
} XrFaceExpressionStatusFB;
```

## Member Descriptions

- `isValid` is an `XrBool32` which indicates that the tracked expression weights are valid.
- `isEyeFollowingBlendshapesValid` is an `XrBool32` which indicates if the 8 expression weights with prefix `XR_FACE_EXPRESSION_EYES_LOOK_*` are valid.

If the returned `isValid` is `XR_FALSE`, then it indicates that the face tracker failed to track or lost track of the face, or the application lost focus, or the consent for face tracking was denied.

If the returned `isValid` is `XR_TRUE`, the runtime **must** return all weights (or all weights except eyes related weights, see `isEyeFollowingBlendshapesValid`).

If the returned `isEyeFollowingBlendshapesValid` is `XR_FALSE`, then it indicates that the eye tracking driving blendshapes with prefix `XR_FACE_EXPRESSION_EYES_LOOK_*` lost track or the consent for eye tracking was denied.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking` extension **must** be enabled prior to using `XrFaceExpressionStatusFB`

### 12.52.6. Example code for obtaining facial expression

The following example code demonstrates how to obtain all weights for facial expression blend shapes.

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized

// Confirm face tracking system support.
XrSystemFaceTrackingPropertiesFB faceTrackingSystemProperties{
    XR_TYPE_SYSTEM_FACE_TRACKING_PROPERTIES_FB};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES,
    &faceTrackingSystemProperties};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
if (!faceTrackingSystemProperties.supportsFaceTracking) {
    // The system does not support face tracking
    return;
}

// Get function pointer for xrCreateFaceTrackerFB.
PFN_xrCreateFaceTrackerFB pfnCreateFaceTrackerFB;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateFaceTrackerFB",
    reinterpret_cast<PFN_xrVoidFunction*>(
    &pfnCreateFaceTrackerFB)));

// Create a face tracker for default set of facial expressions.
XrFaceTrackerFB faceTracker = {};
{
    XrFaceTrackerCreateInfoFB createInfo{XR_TYPE_FACE_TRACKER_CREATE_INFO_FB};
    createInfo.faceExpressionSet = XR_FACE_EXPRESSION_SET_DEFAULT_FB;
    CHK_XR(pfnCreateFaceTrackerFB(session, &createInfo, &faceTracker));
}

// Allocate buffers to receive facial expression data before frame
// loop starts.
float weights[XR_FACE_EXPRESSION_COUNT_FB];
float confidences[XR_FACE_CONFIDENCE_COUNT_FB];

XrFaceExpressionWeightsFB expressionWeights{XR_TYPE_FACE_EXPRESSION_WEIGHTS_FB};
expressionWeights.weightCount = XR_FACE_EXPRESSION_COUNT_FB;
expressionWeights.weights = weights;
```

```

expressionWeights.confidenceCount = XR_FACE_CONFIDENCE_COUNT_FB;
expressionWeights.confidences = confidences;

// Get function pointer for xrGetFaceExpressionWeightsFB.
PFN_xrGetFaceExpressionWeightsFB pfnGetFaceExpressionWeights;
CHK_XR(xrGetInstanceProcAddr(instance, "xrGetFaceExpressionWeightsFB",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnGetFaceExpressionWeights)));

while (1) {
    // ...
    // For every frame in the frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrFaceExpressionInfoFB expressionInfo{XR_TYPE_FACE_EXPRESSION_INFO_FB};
    expressionInfo.time = time;

    CHK_XR(pfnGetFaceExpressionWeights(faceTracker, &expressionInfo,
                                       &expressionWeights));

    if (expressionWeights.status.isValid) {
        for (uint32_t i = 0; i < XR_FACE_EXPRESSION_COUNT_FB; ++i) {
            // weights[i] contains a weight of specific blend shape
        }
    }
}

```

## 12.52.7. Conventions of blend shapes

This extension defines 63 blend shapes for tracking facial expressions.

```

// Provided by XR_FB_face_tracking
typedef enum XrFaceExpressionFB {
    XR_FACE_EXPRESSION_BROW_LOWERER_L_FB = 0,
    XR_FACE_EXPRESSION_BROW_LOWERER_R_FB = 1,
    XR_FACE_EXPRESSION_CHEEK_PUFF_L_FB = 2,
    XR_FACE_EXPRESSION_CHEEK_PUFF_R_FB = 3,
    XR_FACE_EXPRESSION_CHEEK_RAISER_L_FB = 4,
    XR_FACE_EXPRESSION_CHEEK_RAISER_R_FB = 5,
    XR_FACE_EXPRESSION_CHEEK_SUCK_L_FB = 6,
    XR_FACE_EXPRESSION_CHEEK_SUCK_R_FB = 7,
    XR_FACE_EXPRESSION_CHIN_RAISER_B_FB = 8,
    XR_FACE_EXPRESSION_CHIN_RAISER_T_FB = 9,

```

XR\_FACE\_EXPRESSION\_DIMPLER\_L\_FB = 10,  
XR\_FACE\_EXPRESSION\_DIMPLER\_R\_FB = 11,  
XR\_FACE\_EXPRESSION\_EYES\_CLOSED\_L\_FB = 12,  
XR\_FACE\_EXPRESSION\_EYES\_CLOSED\_R\_FB = 13,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_DOWN\_L\_FB = 14,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_DOWN\_R\_FB = 15,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_LEFT\_L\_FB = 16,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_LEFT\_R\_FB = 17,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_RIGHT\_L\_FB = 18,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_RIGHT\_R\_FB = 19,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_UP\_L\_FB = 20,  
XR\_FACE\_EXPRESSION\_EYES\_LOOK\_UP\_R\_FB = 21,  
XR\_FACE\_EXPRESSION\_INNER\_BROW\_RAISER\_L\_FB = 22,  
XR\_FACE\_EXPRESSION\_INNER\_BROW\_RAISER\_R\_FB = 23,  
XR\_FACE\_EXPRESSION\_JAW\_DROP\_FB = 24,  
XR\_FACE\_EXPRESSION\_JAW\_SIDEWAYS\_LEFT\_FB = 25,  
XR\_FACE\_EXPRESSION\_JAW\_SIDEWAYS\_RIGHT\_FB = 26,  
XR\_FACE\_EXPRESSION\_JAW\_THRUST\_FB = 27,  
XR\_FACE\_EXPRESSION\_LID\_TIGHTENER\_L\_FB = 28,  
XR\_FACE\_EXPRESSION\_LID\_TIGHTENER\_R\_FB = 29,  
XR\_FACE\_EXPRESSION\_LIP\_CORNER\_DEPRESSOR\_L\_FB = 30,  
XR\_FACE\_EXPRESSION\_LIP\_CORNER\_DEPRESSOR\_R\_FB = 31,  
XR\_FACE\_EXPRESSION\_LIP\_CORNER\_PULLER\_L\_FB = 32,  
XR\_FACE\_EXPRESSION\_LIP\_CORNER\_PULLER\_R\_FB = 33,  
XR\_FACE\_EXPRESSION\_LIP\_FUNNELER\_LB\_FB = 34,  
XR\_FACE\_EXPRESSION\_LIP\_FUNNELER\_LT\_FB = 35,  
XR\_FACE\_EXPRESSION\_LIP\_FUNNELER\_RB\_FB = 36,  
XR\_FACE\_EXPRESSION\_LIP\_FUNNELER\_RT\_FB = 37,  
XR\_FACE\_EXPRESSION\_LIP\_PRESSOR\_L\_FB = 38,  
XR\_FACE\_EXPRESSION\_LIP\_PRESSOR\_R\_FB = 39,  
XR\_FACE\_EXPRESSION\_LIP\_PUCKER\_L\_FB = 40,  
XR\_FACE\_EXPRESSION\_LIP\_PUCKER\_R\_FB = 41,  
XR\_FACE\_EXPRESSION\_LIP\_STRETCHER\_L\_FB = 42,  
XR\_FACE\_EXPRESSION\_LIP\_STRETCHER\_R\_FB = 43,  
XR\_FACE\_EXPRESSION\_LIP\_SUCK\_LB\_FB = 44,  
XR\_FACE\_EXPRESSION\_LIP\_SUCK\_LT\_FB = 45,  
XR\_FACE\_EXPRESSION\_LIP\_SUCK\_RB\_FB = 46,  
XR\_FACE\_EXPRESSION\_LIP\_SUCK\_RT\_FB = 47,  
XR\_FACE\_EXPRESSION\_LIP\_TIGHTENER\_L\_FB = 48,  
XR\_FACE\_EXPRESSION\_LIP\_TIGHTENER\_R\_FB = 49,  
XR\_FACE\_EXPRESSION\_LIPS\_TOWARD\_FB = 50,  
XR\_FACE\_EXPRESSION\_LOWER\_LIP\_DEPRESSOR\_L\_FB = 51,  
XR\_FACE\_EXPRESSION\_LOWER\_LIP\_DEPRESSOR\_R\_FB = 52,  
XR\_FACE\_EXPRESSION\_MOUTH\_LEFT\_FB = 53,  
XR\_FACE\_EXPRESSION\_MOUTH\_RIGHT\_FB = 54,  
XR\_FACE\_EXPRESSION\_NOSE\_WRINKLER\_L\_FB = 55,  
XR\_FACE\_EXPRESSION\_NOSE\_WRINKLER\_R\_FB = 56,  
XR\_FACE\_EXPRESSION\_OUTER\_BROW\_RAISER\_L\_FB = 57,



```

XR_FACE_EXPRESSION_OUTER_BROW_RAISER_R_FB = 58,
XR_FACE_EXPRESSION_UPPER_LID_RAISER_L_FB = 59,
XR_FACE_EXPRESSION_UPPER_LID_RAISER_R_FB = 60,
XR_FACE_EXPRESSION_UPPER_LIP_RAISER_L_FB = 61,
XR_FACE_EXPRESSION_UPPER_LIP_RAISER_R_FB = 62,
XR_FACE_EXPRESSION_COUNT_FB = 63,
XR_FACE_EXPRESSION_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceExpressionFB;

```

### 12.52.8. Conventions of confidence areas

This extension defines two separate areas of confidence.

```

// Provided by XR_FB_face_tracking
typedef enum XrFaceConfidenceFB {
    XR_FACE_CONFIDENCE_LOWER_FACE_FB = 0,
    XR_FACE_CONFIDENCE_UPPER_FACE_FB = 1,
    XR_FACE_CONFIDENCE_COUNT_FB = 2,
    XR_FACE_CONFIDENCE_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceConfidenceFB;

```

The "upper face" area represents everything above the upper lip, including eye, eyebrows + cheek, and nose. The "lower face" area represents everything under eyes, including mouth, chin + cheek, and nose. Cheek and nose areas contribute to both "upper face" and "lower face" areas.

#### New Object Types

- [XrFaceTrackerFB](#)

#### New Flag Types

#### New Enum Constants

[XrObjectType](#) enumeration is extended with:

- [XR\\_OBJECT\\_TYPE\\_FACE\\_TRACKER\\_FB](#)

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SYSTEM\\_FACE\\_TRACKING\\_PROPERTIES\\_FB](#)
- [XR\\_TYPE\\_FACE\\_TRACKER\\_CREATE\\_INFO\\_FB](#)
- [XR\\_TYPE\\_FACE\\_EXPRESSION\\_INFO\\_FB](#)
- [XR\\_TYPE\\_FACE\\_EXPRESSION\\_WEIGHTS\\_FB](#)

## New Enums

- [XrFaceExpressionFB](#)
- [XrFaceExpressionSetFB](#)
- [XrFaceConfidenceFB](#)

## New Structures

- [XrSystemFaceTrackingPropertiesFB](#)
- [XrFaceTrackerCreateInfoFB](#)
- [XrFaceExpressionInfoFB](#)
- [XrFaceExpressionStatusFB](#)
- [XrFaceExpressionWeightsFB](#)

## New Functions

- [xrCreateFaceTrackerFB](#)
- [xrDestroyFaceTrackerFB](#)
- [xrGetFaceExpressionWeightsFB](#)

## Issues

## Version History

- Revision 1, 2022-07-15 (Igor Tceglevskii)
  - Initial extension description

# 12.53. XR\_FB\_face\_tracking2

## Name String

`XR_FB_face_tracking2`

## Extension Type

Instance extension

## Registered Extension Number

288

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2023-10-06

## IP Status

No known IP claims.

## Contributors

Jaebong Lee, Meta  
Dikpal Reddy, Meta  
Igor Tceglevskii, Meta  
Bill Orr, Meta  
Scott Ramsby, Meta

### 12.53.1. Overview

This extension enables applications to get weights of blend shapes. It also enables applications to render facial expressions in XR experiences.

It is recommended to choose this extension over the [XR\\_FB\\_face\\_tracking](#) extension, if it is supported by the runtime, because this extension provides the following two additional capabilities to the application:

- This extension provides additional seven blend shapes that estimate tongue movement.
- This extension allows an application and the runtime to communicate about the data sources that are used to estimate facial expression in a cooperative manner.

Face tracking data is sensitive personal information and is closely linked to personal privacy and integrity. Applications storing or transferring face tracking data **should** always ask the user for active and specific acceptance to do so.

If the runtime supports a permission system to control application access to the face tracker, then the runtime **must** set the `isValid` field to `XR_FALSE` on the supplied [XrFaceExpressionWeights2FB](#) structure until the user allows the application to access the face tracker. When the application access has been allowed, the runtime **should** set `isValid` on the supplied [XrFaceExpressionWeights2FB](#) structure to `XR_TRUE`.

Some permission systems **may** control access to the eye tracking separately from access to the face tracking, even though the eyes are part of the face. In case the user denied tracking of the eyes, yet, allowed tracking of the face, then the runtime **must** set the `isEyeFollowingBlendshapesValid` field to `XR_FALSE` on the supplied [XrFaceExpressionWeights2FB](#) for indicating that eye tracking data is not available, but at the same time **may** set the `isValid` field to `XR_TRUE` on the supplied [XrFaceExpressionWeights2FB](#) for indicating that another part of the face is tracked properly.

## 12.53.2. Inspect system capability

```
// Provided by XR_FB_face_tracking2
typedef struct XrSystemFaceTrackingProperties2FB {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsVisualFaceTracking;
    XrBool32           supportsAudioFaceTracking;
} XrSystemFaceTrackingProperties2FB;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsVisualFaceTracking` is an [XrBool32](#), indicating if the current system is capable of receiving face tracking input that is estimated based on visual data source.
- `supportsAudioFaceTracking` is an [XrBool32](#), indicating if the current system is capable of receiving face tracking input that is estimated based on audio data source.

An application **can** inspect whether the system is capable of receiving face tracking input by extending the [XrSystemProperties](#) with [XrSystemFaceTrackingProperties2FB](#) structure when calling [xrGetSystemProperties](#).

If an application calls [xrCreateFaceTracker2FB](#) only with unsupported [XrFaceTrackerCreateInfo2FB::requestedDataSources](#), the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateFaceTracker2FB](#). For example, if an application calls [xrCreateFaceTracker2FB](#) only with `XR_FACE_TRACKING_DATA_SOURCE2_AUDIO_FB` in [XrFaceTrackerCreateInfo2FB::requestedDataSources](#) when the runtime returns `XR_FALSE` for `supportsAudioFaceTracking`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateFaceTracker2FB](#).

### Valid Usage (Implicit)

- The [XR\\_FB\\_face\\_tracking2](#) extension **must** be enabled prior to using [XrSystemFaceTrackingProperties2FB](#)
- `type` **must** be `XR_TYPE_SYSTEM_FACE_TRACKING_PROPERTIES2_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.53.3. Create a face tracker handle

The `XrFaceTracker2FB` handle represents the resources for face tracking.

```
// Provided by XR_FB_face_tracking2
XR_DEFINE_HANDLE(XrFaceTracker2FB)
```

This handle is used to obtain blend shapes using the `xrGetFaceExpressionWeights2FB` function.

The `xrCreateFaceTracker2FB` function is defined as:

```
// Provided by XR_FB_face_tracking2
XrResult xrCreateFaceTracker2FB(
    XrSession session,
    const XrFaceTrackerCreateInfo2FB* createInfo,
    XrFaceTracker2FB* faceTracker);
```

#### Parameter Descriptions

- `session` is an `XrSession` in which the face tracker will be active.
- `createInfo` is the `XrFaceTrackerCreateInfo2FB` used to specify the face tracker.
- `faceTracker` is the returned `XrFaceTracker2FB` handle.

An application **can** create an `XrFaceTracker2FB` handle using `xrCreateFaceTracker2FB` function.

If the system does not support face tracking, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateFaceTracker2FB`. In this case, the runtime **must** return `XR_FALSE` for both `XrSystemFaceTrackingProperties2FB::supportsVisualFaceTracking` and `XrSystemFaceTrackingProperties2FB::supportsAudioFaceTracking` when the function `xrGetSystemProperties` is called, so that the application **can** avoid creating a face tracker.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking2` extension **must** be enabled prior to calling `xrCreateFaceTracker2FB`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrFaceTrackerCreateInfo2FB` structure
- `faceTracker` **must** be a pointer to an `XrFaceTracker2FB` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrFaceTrackerCreateInfo2FB` structure is described as follows:

```
// Provided by XR_FB_face_tracking2
typedef struct XrFaceTrackerCreateInfo2FB {
    XrStructureType          type;
    const void*              next;
    XrFaceExpressionSet2FB   faceExpressionSet;
    uint32_t                 requestedDataSourceCount;
    XrFaceTrackingDataSource2FB* requestedDataSources;
} XrFaceTrackerCreateInfo2FB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `faceExpressionSet` is an [XrFaceExpressionSet2FB](#) that describes the set of blend shapes to retrieve.
- `requestedDataSourceCount` is the number of elements in the `requestedDataSources` array.
- `requestedDataSources` is an array of [XrFaceTrackingDataSource2FB](#) that the application accepts. The order of values in the array has no significance.

The [XrFaceTrackerCreateInfo2FB](#) structure describes the information to create an [XrFaceTracker2FB](#) handle.

Runtimes **may** support a variety of data sources for estimations of facial expression, and some runtimes and devices **may** use data from multiple data sources. The application tells the runtime all data sources that the runtime **may** use to provide facial expressions for the application.

Because the device setting **may** change during a running session, the runtime **may** return a valid [XrFaceTracker2FB](#) handle even if the device is unable to estimate facial expression using the data sources requested by the application's call to [xrCreateFaceTracker2FB](#). The runtime **must** instead return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateFaceTracker2FB](#), if for example the runtime believes it will never be able to satisfy the request.

If `requestedDataSourceCount` is `0`, the runtime **may** choose any supported data source, preferably one that is more expressive than the others.

If any value in `requestedDataSources` is duplicated the runtime **must** return `XR_ERROR_VALIDATION_FAILURE` from the call to [xrCreateFaceTracker2FB](#).

## Valid Usage (Implicit)

- The `XR_FB_face_tracking2` extension **must** be enabled prior to using [XrFaceTrackerCreateInfo2FB](#)
- `type` **must** be `XR_TYPE_FACE_TRACKER_CREATE_INFO2_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `faceExpressionSet` **must** be a valid [XrFaceExpressionSet2FB](#) value
- If `requestedDataSourceCount` is not `0`, `requestedDataSources` **must** be a pointer to an array of `requestedDataSourceCount` [XrFaceTrackingDataSource2FB](#) values

The [XrFaceExpressionSet2FB](#) enum describes the set of blend shapes of a facial expression to track when creating an [XrFaceTracker2FB](#).

```
// Provided by XR_FB_face_tracking2
typedef enum XrFaceExpressionSet2FB {
    XR_FACE_EXPRESSION_SET2_DEFAULT_FB = 0,
    XR_FACE_EXPRESSION_SET_2FB_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceExpressionSet2FB;
```

### Enumerant Descriptions

- [XR\\_FACE\\_EXPRESSION\\_SET2\\_DEFAULT\\_FB](#) — indicates that the created [XrFaceTracker2FB](#) tracks the set of blend shapes described by [XrFaceExpression2FB](#) enum, i.e. the [xrGetFaceExpressionWeights2FB](#) function returns an array of blend shapes with the count of [XR\\_FACE\\_EXPRESSION2\\_COUNT\\_FB](#) and **can** be indexed using [XrFaceExpression2FB](#).

The [XrFaceTrackingDataSource2FB](#) enumeration is defined as:

```
// Provided by XR_FB_face_tracking2
typedef enum XrFaceTrackingDataSource2FB {
    XR_FACE_TRACKING_DATA_SOURCE2_VISUAL_FB = 0,
    XR_FACE_TRACKING_DATA_SOURCE2_AUDIO_FB = 1,
    XR_FACE_TRACKING_DATA_SOURCE_2FB_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceTrackingDataSource2FB;
```

### Enumerant Descriptions

- [XR\\_FACE\\_TRACKING\\_DATA\\_SOURCE2\\_VISUAL\\_FB](#) - This value indicates that the face tracking data source supports using visual data to estimate facial expression. The runtime **may** also use audio to further improve the quality of the tracking.
- [XR\\_FACE\\_TRACKING\\_DATA\\_SOURCE2\\_AUDIO\\_FB](#) - This value indicates that the face tracking data source supports using audio data to estimate facial expression. The runtime **must** not use visual data for this data source.

## 12.53.4. Delete a face tracker handle

The [xrDestroyFaceTracker2FB](#) function is defined as:



```
// Provided by XR_FB_face_tracking2
XrResult xrDestroyFaceTracker2FB(
    XrFaceTracker2FB          faceTracker);
```

## Parameter Descriptions

- `faceTracker` is an `XrFaceTracker2FB` previously created by `xrCreateFaceTracker2FB`.

The `xrDestroyFaceTracker2FB` function releases the `faceTracker` and the underlying resources when face tracking experience is over.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking2` extension **must** be enabled prior to calling `xrDestroyFaceTracker2FB`
- `faceTracker` **must** be a valid `XrFaceTracker2FB` handle

## Thread Safety

- Access to `faceTracker`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## 12.53.5. Obtain facial expressions

The `xrGetFaceExpressionWeights2FB` function is defined as:

```
// Provided by XR_FB_face_tracking2
XrResult xrGetFaceExpressionWeights2FB(
    XrFaceTracker2FB          faceTracker,
    const XrFaceExpressionInfo2FB* expressionInfo,
    XrFaceExpressionWeights2FB* expressionWeights);
```

## Parameter Descriptions

- `faceTracker` is an `XrFaceTracker2FB` previously created by `xrCreateFaceTracker2FB`.
- `expressionInfo` is a pointer to `XrFaceExpressionInfo2FB` describing information to obtain face expression.
- `expressionWeights` is a pointer to `XrFaceExpressionWeights2FB` receiving the returned facial expression weights.

The `xrGetFaceExpressionWeights2FB` function return blend shapes of facial expression at a given time.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking2` extension **must** be enabled prior to calling `xrGetFaceExpressionWeights2FB`
- `faceTracker` **must** be a valid `XrFaceTracker2FB` handle
- `expressionInfo` **must** be a pointer to a valid `XrFaceExpressionInfo2FB` structure
- `expressionWeights` **must** be a pointer to an `XrFaceExpressionWeights2FB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrFaceExpressionInfo2FB` structure is defined as:

```
// Provided by XR_FB_face_tracking2
typedef struct XrFaceExpressionInfo2FB {
    XrStructureType    type;
    const void*        next;
    XrTime              time;
} XrFaceExpressionInfo2FB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `time` is an `XrTime` at which the facial expression weights are requested.

The `XrFaceExpressionInfo2FB` structure describes the information to obtain facial expression. The application **should** pass a time equal to the predicted display time for the rendered frame. The system **must** employ appropriate modeling to provide expressions for this time.

## Valid Usage (Implicit)

- The `XR_FB_face_tracking2` extension **must** be enabled prior to using `XrFaceExpressionInfo2FB`
- `type` **must** be `XR_TYPE_FACE_EXPRESSION_INFO2_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

The `XrFaceExpressionWeights2FB` structure is defined as:

```
// Provided by XR_FB_face_tracking2
typedef struct XrFaceExpressionWeights2FB {
    XrStructureType          type;
    void*                    next;
    uint32_t                 weightCount;
    float*                   weights;
    uint32_t                 confidenceCount;
    float*                   confidences;
    XrBool32                 isValid;
    XrBool32                 isEyeFollowingBlendshapesValid;
    XrFaceTrackingDataSource2FB dataSource;
    XrTime                   time;
} XrFaceExpressionWeights2FB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `weightCount` is a `uint32_t` describing the count of elements in `weights` array.
- `weights` is a pointer to an application-allocated array of `float` that will be filled with weights of facial expression blend shapes.
- `confidenceCount` is a `uint32_t` describing the count of elements in `confidences` array.
- `confidences` is a pointer to an application-allocated array of `float` that will be filled with confidence of tracking specific parts of a face.
- `isValid` is an [XrBool32](#) which indicates that the tracked expression weights are valid.
- `isEyeFollowingBlendshapesValid` is an [XrBool32](#) which indicates if the 8 expression weights with prefix `XR_FACE_EXPRESSION2_EYES_LOOK_*` are valid.
- `dataSource` is an [XrFaceTrackingDataSource2FB](#) which indicates the data source that was used to estimate the facial expression.
- `time` is an [XrTime](#) time at which the returned expression weights are tracked or extrapolated to. Equals the time at which the expression weights were requested if the extrapolating at the time was successful.

[XrFaceExpressionWeights2FB](#) structure returns the facial expression.

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `weightCount` is not equal to the number of blend shapes defined by the [XrFaceExpressionSet2FB](#) used to create the [XrFaceTracker2FB](#).

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `confidenceCount` is not equal to the number of confidence areas defined by the [XrFaceExpressionSet2FB](#) used to create the [XrFaceTracker2FB](#).

The runtime **must** return `weights` representing the weights of blend shapes of current facial expression.

The runtime **must** update the `weights` array ordered so that the application **can** index elements using the corresponding facial expression enum (e.g. [XrFaceExpression2FB](#)) as described by [XrFaceExpressionSet2FB](#) when creating the [XrFaceTracker2FB](#). For example, when the [XrFaceTracker2FB](#) is created with `XR_FACE_EXPRESSION_SET2_DEFAULT_FB`, the application sets the `weightCount` to `XR_FACE_EXPRESSION2_COUNT_FB`, and the runtime **must** fill the `weights` array ordered so that it **can** be indexed by the [XrFaceExpression2FB](#) enum.

The runtime **must** update the `confidences` array ordered so that the application **can** index elements using the corresponding confidence area enum (e.g. [XrFaceConfidence2FB](#)) as described by [XrFaceExpressionSet2FB](#) when creating the [XrFaceTracker2FB](#). For example, when the

`XrFaceTracker2FB` is created with `XR_FACE_EXPRESSION_SET2_DEFAULT_FB`, the application sets the `confidenceCount` to `XR_FACE_CONFIDENCE2_COUNT_FB`, and the runtime **must** fill the `confidences` array ordered so that it **can** be indexed by the `XrFaceConfidence2FB` enum.

The runtime **must** set `isValid` to `XR_FALSE` and it **must** also set all elements of `weights` to zero, if one of the following is true:

- the face tracker failed to track or lost track of the face
- the application lost focus
- the consent for face tracking was denied
- the runtime is unable to estimate facial expression from the data sources specified when `xrCreateFaceTracker2FB` function was called

If the returned `isValid` is `XR_TRUE`, the runtime **must** return all weights (or all weights except eyes related weights, see `isEyeFollowingBlendshapesValid`).

The runtime **must** set `isEyeFollowingBlendshapesValid` to `XR_FALSE` and it **must** also set 8 expression weights with prefix `XR_FACE_EXPRESSION2_EYES_LOOK_*` to zero, if one of the following is true:

- the eye tracking driving blendshapes with prefix `XR_FACE_EXPRESSION2_EYES_LOOK_*` lost track
- the consent for eye tracking was denied

### Valid Usage (Implicit)

- The `XR_FB_face_tracking2` extension **must** be enabled prior to using `XrFaceExpressionWeights2FB`
- `type` **must** be `XR_TYPE_FACE_EXPRESSION_WEIGHTS2_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `weights` **must** be a pointer to an array of `weightCount` float values
- `confidences` **must** be a pointer to an array of `confidenceCount` float values
- `dataSource` **must** be a valid `XrFaceTrackingDataSource2FB` value
- The `weightCount` parameter **must** be greater than 0
- The `confidenceCount` parameter **must** be greater than 0

## 12.53.6. Example code for obtaining facial expression

The following example code demonstrates how to obtain all weights for facial expression blend shapes.

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
```

```

XrSession session; // previously initialized

// Confirm face tracking system support.
XrSystemFaceTrackingProperties2FB faceTrackingSystemProperties{
    XR_TYPE_SYSTEM_FACE_TRACKING_PROPERTIES2_FB};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES,
    &faceTrackingSystemProperties};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
if (!faceTrackingSystemProperties.supportsVisualFaceTracking &&
    !faceTrackingSystemProperties.supportsAudioFaceTracking) {
    // The system does not support face tracking
    return;
}

// Get function pointer for xrCreateFaceTracker2FB.
PFN_xrCreateFaceTracker2FB pfnCreateFaceTracker2FB;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateFaceTracker2FB",
    reinterpret_cast<PFN_xrVoidFunction*>(
        &pfnCreateFaceTracker2FB)));

// Create a face tracker for default set of facial expressions.
XrFaceTracker2FB faceTracker = {};
{
    XrFaceTrackerCreateInfo2FB createInfo{XR_TYPE_FACE_TRACKER_CREATE_INFO2_FB};
    createInfo.faceExpressionSet = XR_FACE_EXPRESSION_SET2_DEFAULT_FB;
    // This tells the runtime that the application can take
    // facial expression from any of two data sources.
    createInfo.requestedDataSourceCount = 2;
    XrFaceTrackingDataSource2FB dataSources[2] = {
        XR_FACE_TRACKING_DATA_SOURCE2_VISUAL_FB,
        XR_FACE_TRACKING_DATA_SOURCE2_AUDIO_FB};
    createInfo.requestedDataSources = dataSources;
    CHK_XR(pfnCreateFaceTracker2FB(session, &createInfo, &faceTracker));
}

// Allocate buffers to receive facial expression data before frame
// loop starts.
float weights[XR_FACE_EXPRESSION2_COUNT_FB];
float confidences[XR_FACE_CONFIDENCE2_COUNT_FB];

XrFaceExpressionWeights2FB expressionWeights{XR_TYPE_FACE_EXPRESSION_WEIGHTS2_FB};
expressionWeights.weightCount = XR_FACE_EXPRESSION2_COUNT_FB;
expressionWeights.weights = weights;
expressionWeights.confidenceCount = XR_FACE_CONFIDENCE2_COUNT_FB;
expressionWeights.confidences = confidences;

// Get function pointer for xrGetFaceExpressionWeights2FB.
PFN_xrGetFaceExpressionWeights2FB pfnGetFaceExpressionWeights;

```

```

CHK_XR(xrGetInstanceProcAddr(instance, "xrGetFaceExpressionWeights2FB",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                             &pfnGetFaceExpressionWeights)));
while (1) {
    // ...
    // For every frame in the frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrFaceExpressionInfo2FB expressionInfo{XR_TYPE_FACE_EXPRESSION_INFO2_FB};
    expressionInfo.time = time;

    CHK_XR(pfnGetFaceExpressionWeights(faceTracker, &expressionInfo,
    &expressionWeights));

    if (expressionWeights.isValid) {
        // If you want to do something depending on the data source.
        if (expressionWeights.dataSource == XR_FACE_TRACKING_DATA_SOURCE2_VISUAL_FB) {
            // do something when visual or audiovisual data source was used.
        } else if (expressionWeights.dataSource ==
XR_FACE_TRACKING_DATA_SOURCE2_AUDIO_FB) {
            // do something when audio data source was used.
        }

        for (uint32_t i = 0; i < XR_FACE_EXPRESSION2_COUNT_FB; ++i) {
            // weights[i] contains a weight of specific blend shape
        }
    }
}
}

```

### 12.53.7. Conventions of blend shapes

This extension defines 70 blend shapes for tracking facial expressions.

```

// Provided by XR_FB_face_tracking2
typedef enum XrFaceExpression2FB {
    XR_FACE_EXPRESSION2_BROW_LOWERER_L_FB = 0,
    XR_FACE_EXPRESSION2_BROW_LOWERER_R_FB = 1,
    XR_FACE_EXPRESSION2_CHEEK_PUFF_L_FB = 2,
    XR_FACE_EXPRESSION2_CHEEK_PUFF_R_FB = 3,
    XR_FACE_EXPRESSION2_CHEEK_RAISER_L_FB = 4,
    XR_FACE_EXPRESSION2_CHEEK_RAISER_R_FB = 5,
    XR_FACE_EXPRESSION2_CHEEK_SUCK_L_FB = 6,

```

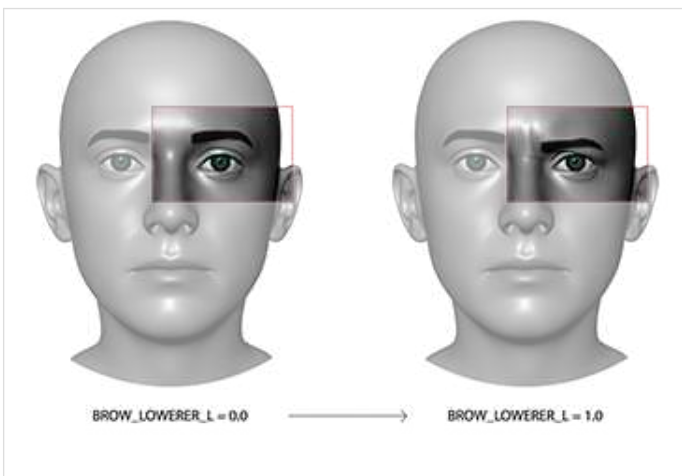


XR\_FACE\_EXPRESSION2\_CHEEK\_SUCK\_R\_FB = 7,  
XR\_FACE\_EXPRESSION2\_CHIN\_RAISER\_B\_FB = 8,  
XR\_FACE\_EXPRESSION2\_CHIN\_RAISER\_T\_FB = 9,  
XR\_FACE\_EXPRESSION2\_DIMPLER\_L\_FB = 10,  
XR\_FACE\_EXPRESSION2\_DIMPLER\_R\_FB = 11,  
XR\_FACE\_EXPRESSION2\_EYES\_CLOSED\_L\_FB = 12,  
XR\_FACE\_EXPRESSION2\_EYES\_CLOSED\_R\_FB = 13,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_DOWN\_L\_FB = 14,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_DOWN\_R\_FB = 15,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_LEFT\_L\_FB = 16,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_LEFT\_R\_FB = 17,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_RIGHT\_L\_FB = 18,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_RIGHT\_R\_FB = 19,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_UP\_L\_FB = 20,  
XR\_FACE\_EXPRESSION2\_EYES\_LOOK\_UP\_R\_FB = 21,  
XR\_FACE\_EXPRESSION2\_INNER\_BROW\_RAISER\_L\_FB = 22,  
XR\_FACE\_EXPRESSION2\_INNER\_BROW\_RAISER\_R\_FB = 23,  
XR\_FACE\_EXPRESSION2\_JAW\_DROP\_FB = 24,  
XR\_FACE\_EXPRESSION2\_JAW\_SIDEWAYS\_LEFT\_FB = 25,  
XR\_FACE\_EXPRESSION2\_JAW\_SIDEWAYS\_RIGHT\_FB = 26,  
XR\_FACE\_EXPRESSION2\_JAW\_THRUST\_FB = 27,  
XR\_FACE\_EXPRESSION2\_LID\_TIGHTENER\_L\_FB = 28,  
XR\_FACE\_EXPRESSION2\_LID\_TIGHTENER\_R\_FB = 29,  
XR\_FACE\_EXPRESSION2\_LIP\_CORNER\_DEPRESSOR\_L\_FB = 30,  
XR\_FACE\_EXPRESSION2\_LIP\_CORNER\_DEPRESSOR\_R\_FB = 31,  
XR\_FACE\_EXPRESSION2\_LIP\_CORNER\_PULLER\_L\_FB = 32,  
XR\_FACE\_EXPRESSION2\_LIP\_CORNER\_PULLER\_R\_FB = 33,  
XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_LB\_FB = 34,  
XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_LT\_FB = 35,  
XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_RB\_FB = 36,  
XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_RT\_FB = 37,  
XR\_FACE\_EXPRESSION2\_LIP\_PRESSOR\_L\_FB = 38,  
XR\_FACE\_EXPRESSION2\_LIP\_PRESSOR\_R\_FB = 39,  
XR\_FACE\_EXPRESSION2\_LIP\_PUCKER\_L\_FB = 40,  
XR\_FACE\_EXPRESSION2\_LIP\_PUCKER\_R\_FB = 41,  
XR\_FACE\_EXPRESSION2\_LIP\_STRETCHER\_L\_FB = 42,  
XR\_FACE\_EXPRESSION2\_LIP\_STRETCHER\_R\_FB = 43,  
XR\_FACE\_EXPRESSION2\_LIP\_SUCK\_LB\_FB = 44,  
XR\_FACE\_EXPRESSION2\_LIP\_SUCK\_LT\_FB = 45,  
XR\_FACE\_EXPRESSION2\_LIP\_SUCK\_RB\_FB = 46,  
XR\_FACE\_EXPRESSION2\_LIP\_SUCK\_RT\_FB = 47,  
XR\_FACE\_EXPRESSION2\_LIP\_TIGHTENER\_L\_FB = 48,  
XR\_FACE\_EXPRESSION2\_LIP\_TIGHTENER\_R\_FB = 49,  
XR\_FACE\_EXPRESSION2\_LIPS\_TOWARD\_FB = 50,  
XR\_FACE\_EXPRESSION2\_LOWER\_LIP\_DEPRESSOR\_L\_FB = 51,  
XR\_FACE\_EXPRESSION2\_LOWER\_LIP\_DEPRESSOR\_R\_FB = 52,  
XR\_FACE\_EXPRESSION2\_MOUTH\_LEFT\_FB = 53,  
XR\_FACE\_EXPRESSION2\_MOUTH\_RIGHT\_FB = 54,

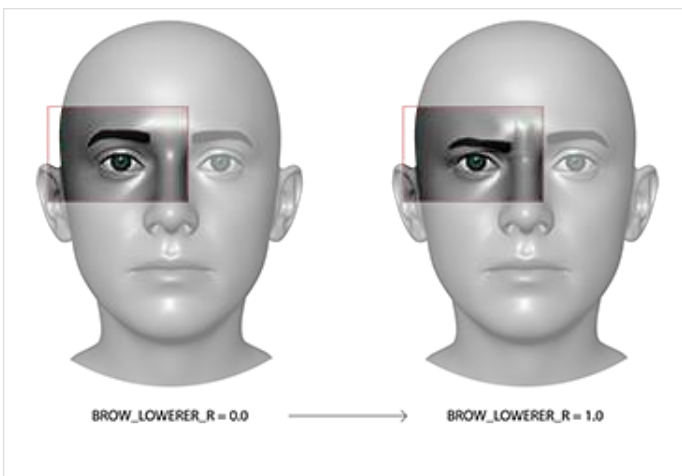
```

XR_FACE_EXPRESSION2_NOSE_WRINKLER_L_FB = 55,
XR_FACE_EXPRESSION2_NOSE_WRINKLER_R_FB = 56,
XR_FACE_EXPRESSION2_OUTER_BROW_RAISER_L_FB = 57,
XR_FACE_EXPRESSION2_OUTER_BROW_RAISER_R_FB = 58,
XR_FACE_EXPRESSION2_UPPER_LID_RAISER_L_FB = 59,
XR_FACE_EXPRESSION2_UPPER_LID_RAISER_R_FB = 60,
XR_FACE_EXPRESSION2_UPPER_LIP_RAISER_L_FB = 61,
XR_FACE_EXPRESSION2_UPPER_LIP_RAISER_R_FB = 62,
XR_FACE_EXPRESSION2_TONGUE_TIP_INTERDENTAL_FB = 63,
XR_FACE_EXPRESSION2_TONGUE_TIP_ALVEOLAR_FB = 64,
XR_FACE_EXPRESSION2_TONGUE_FRONT_DORSAL_PALATE_FB = 65,
XR_FACE_EXPRESSION2_TONGUE_MID_DORSAL_PALATE_FB = 66,
XR_FACE_EXPRESSION2_TONGUE_BACK_DORSAL_VELAR_FB = 67,
XR_FACE_EXPRESSION2_TONGUE_OUT_FB = 68,
XR_FACE_EXPRESSION2_TONGUE_RETREAT_FB = 69,
XR_FACE_EXPRESSION2_COUNT_FB = 70,
XR_FACE_EXPRESSION_2FB_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceExpression2FB;

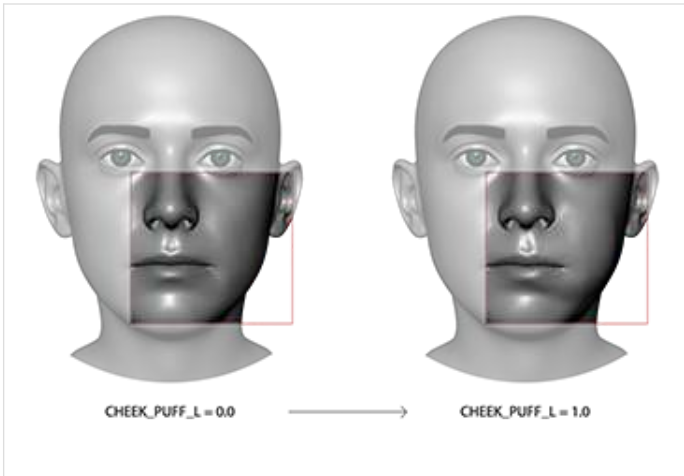
```



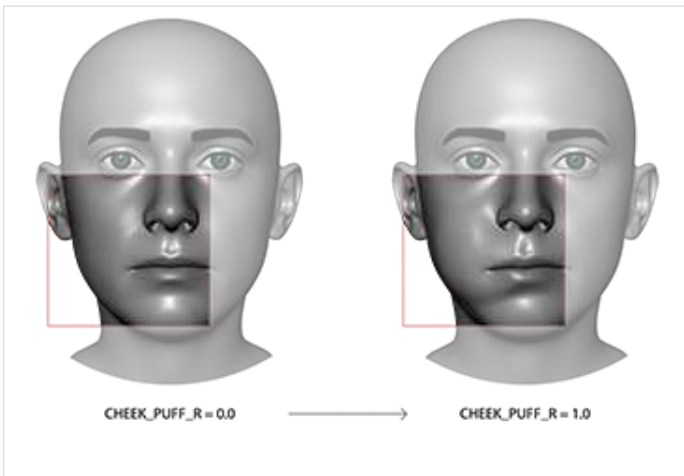
**XR\_FACE\_EXPRESSION2\_BROW\_LOWERER\_L\_FB** knits and lowers the left brow area and lowers central forehead.



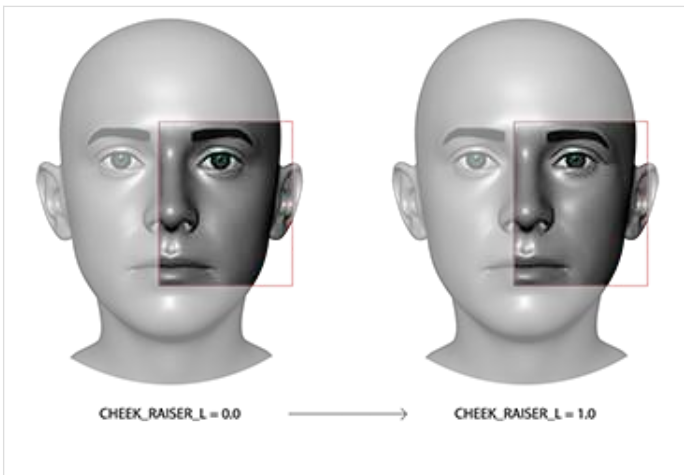
**XR\_FACE\_EXPRESSION2\_BROW\_LOWERER\_R\_FB** knits and lowers the right brow area and lowers central forehead.



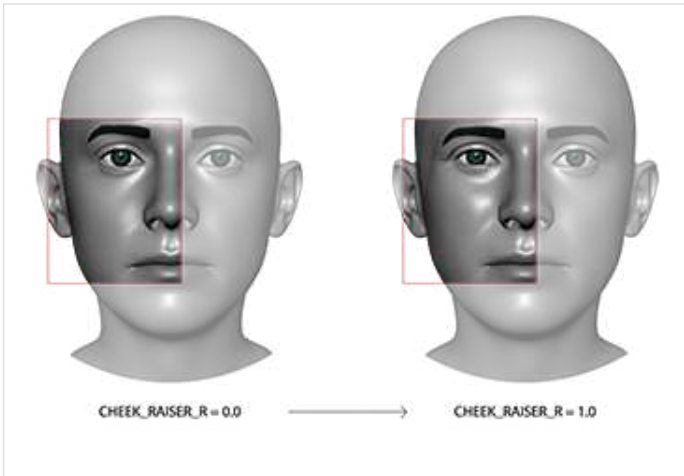
`XR_FACE_EXPRESSION2_CHEEK_PUFF_L_FB` fills the left cheek with air causing them to round and extend outward.



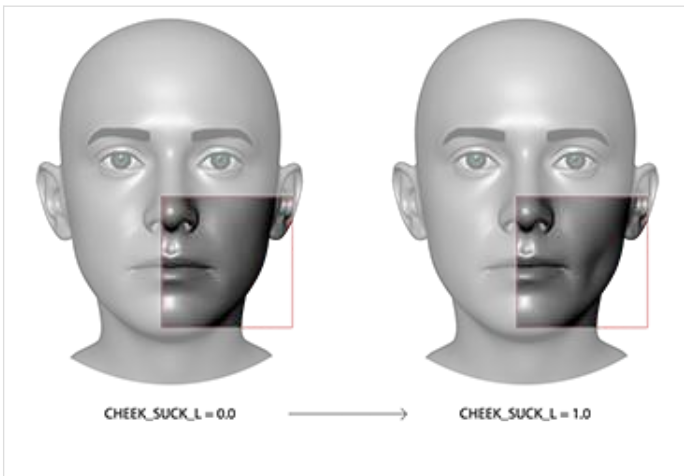
`XR_FACE_EXPRESSION2_CHEEK_PUFF_R_FB` fills the right cheek with air causing them to round and extend outward.



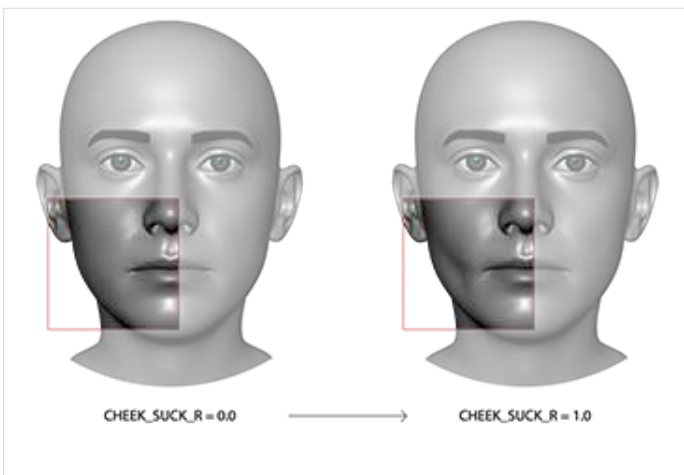
`XR_FACE_EXPRESSION2_CHEEK_RAISER_L_FB` tightens the outer rings of the left eye orbit and squeezes the lateral left eye corners.



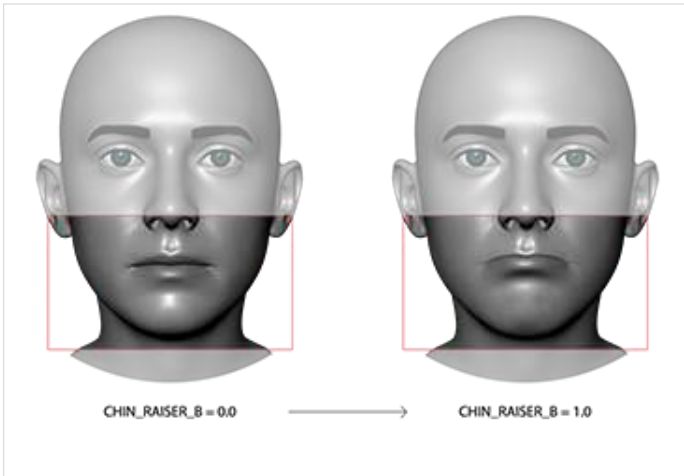
`XR_FACE_EXPRESSION2_CHEEK_RAISER_R_FB` tightens the outer rings of the right eye orbit and squeezes the lateral right eye corners.



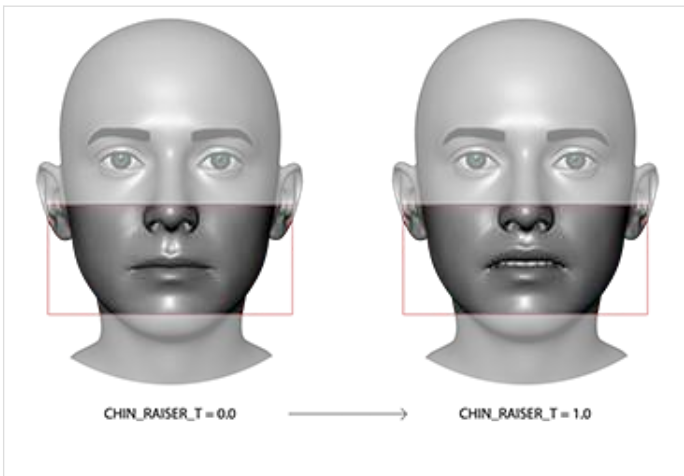
`XR_FACE_EXPRESSION2_CHEEK_SUCK_L_FB` sucks the left cheek inward and against the teeth to create a hollow effect in the cheek.



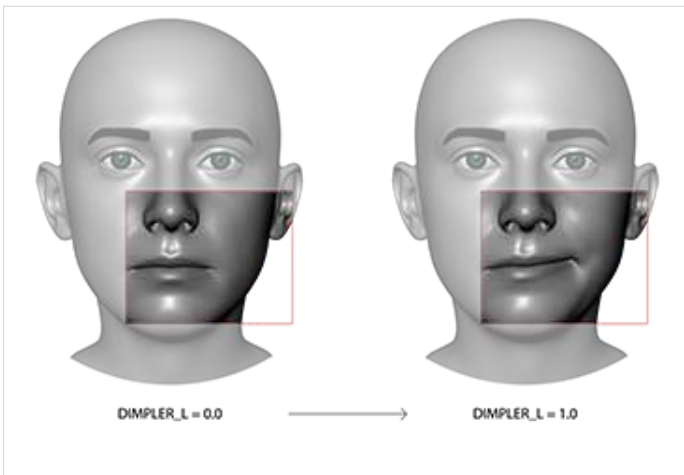
`XR_FACE_EXPRESSION2_CHEEK_SUCK_R_FB` sucks the right cheek inward and against the teeth to create a hollow effect in the cheek.



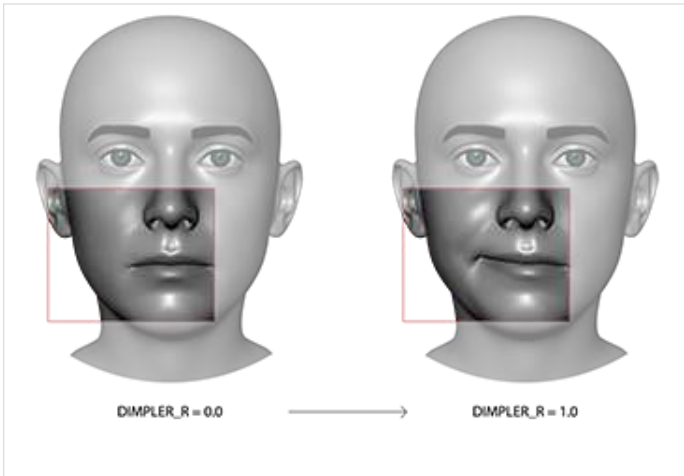
`XR_FACE_EXPRESSION2_CHIN_RAISER_B_FB` pushes the skin of the chin and the lower lip upward.



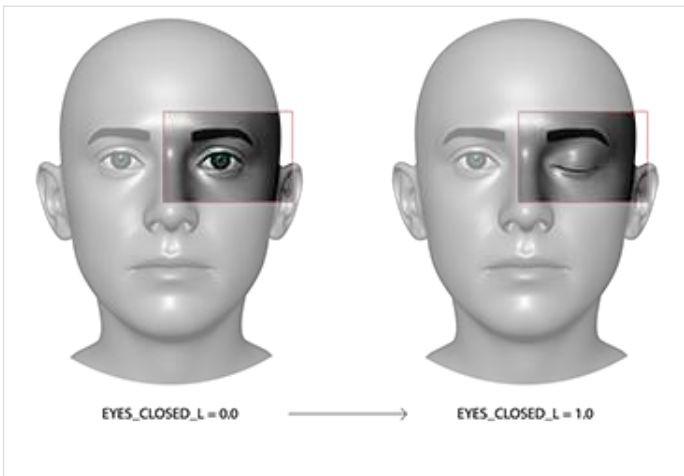
`XR_FACE_EXPRESSION2_CHIN_RAISER_T_FB` pushes up the top lip. This is induced by the upward force from `XR_FACE_EXPRESSION2_CHIN_RAISER_B_FB`.



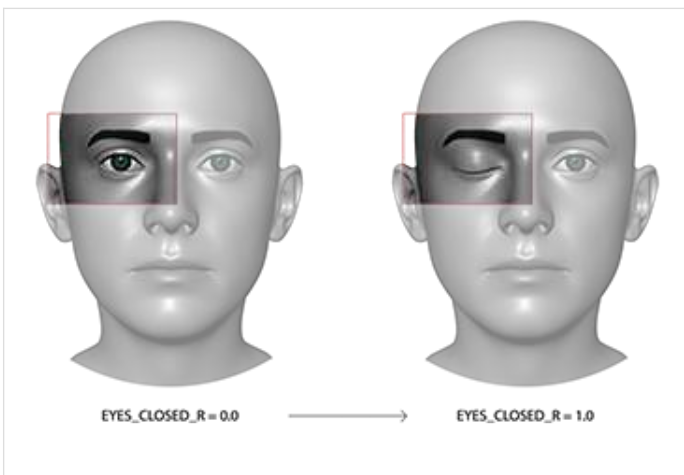
`XR_FACE_EXPRESSION2_DIMPLER_L_FB` pinches the left lip corner against the teeth, drawing them slightly backward and often upward in the process.



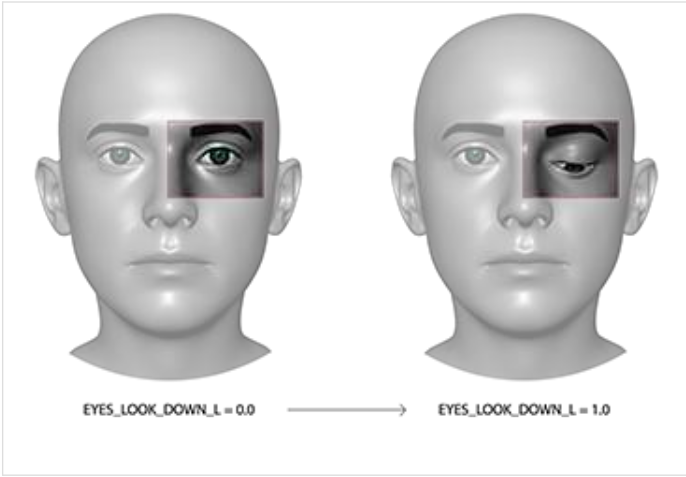
**XR\_FACE\_EXPRESSION2\_DIMPLER\_R\_FB** pinches the right lip corner against the teeth, drawing them slightly backward and often upward in the process.



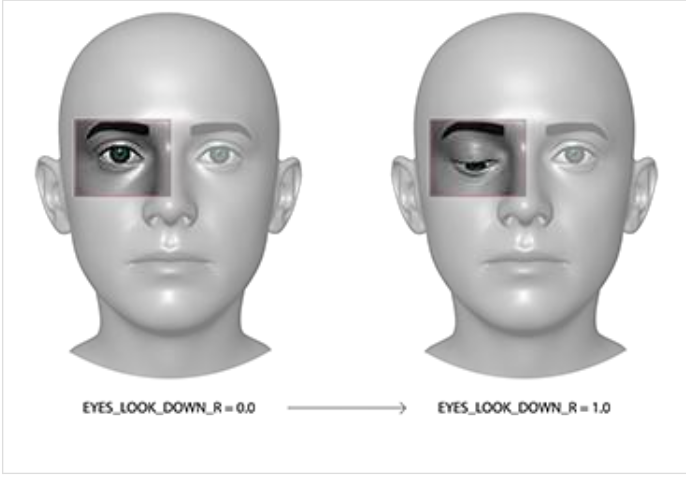
**XR\_FACE\_EXPRESSION2\_EYES\_CLOSED\_L\_FB** lowers the top eyelid to cover the left eye.



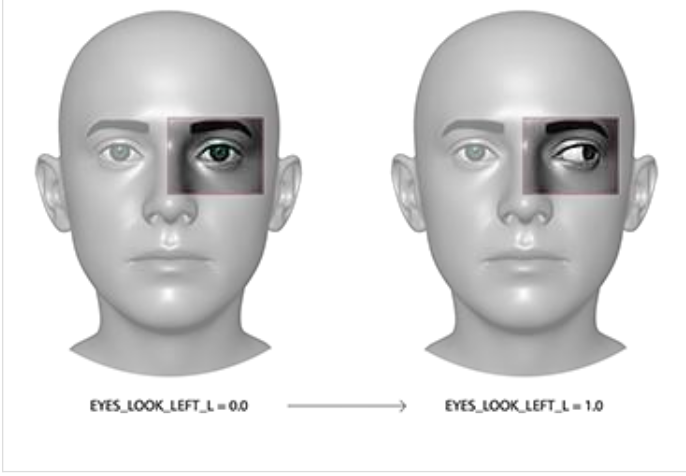
**XR\_FACE\_EXPRESSION2\_EYES\_CLOSED\_R\_FB** lowers the top eyelid to cover the right eye.



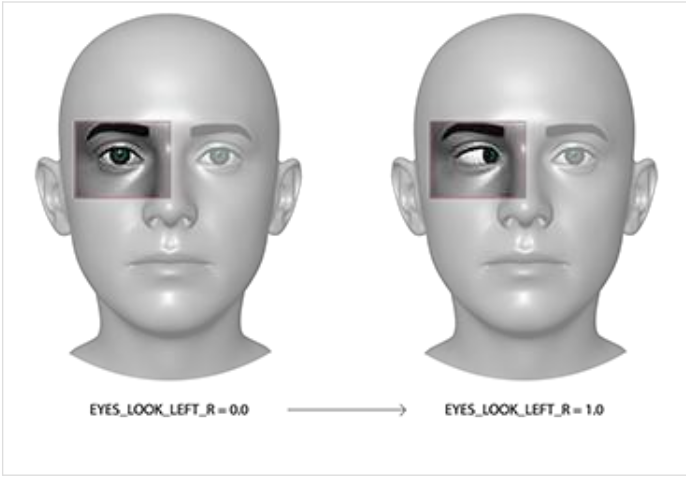
`XR_FACE_EXPRESSION2_EYES_LOOK_DOWN_L_FB` moves the left eyelid consistent with downward gaze.



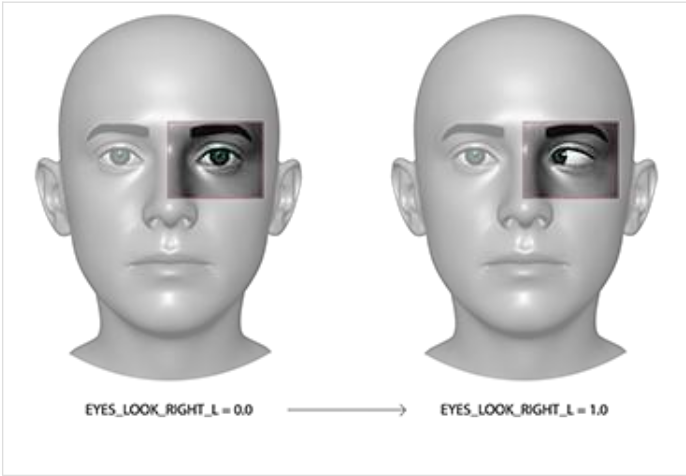
`XR_FACE_EXPRESSION2_EYES_LOOK_DOWN_R_FB` moves the right eyelid consistent with downward gaze.



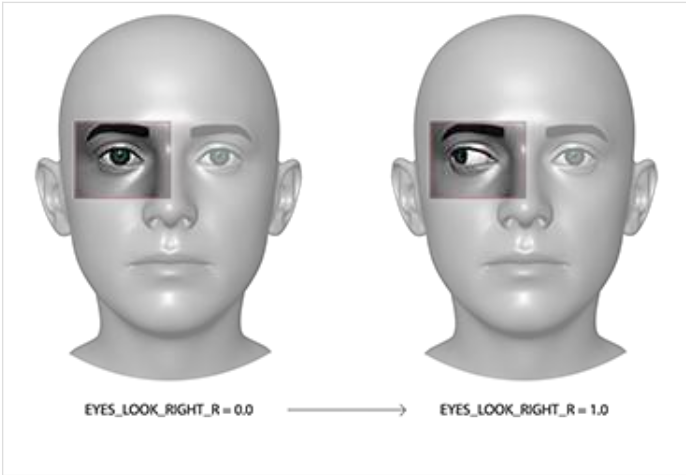
`XR_FACE_EXPRESSION2_EYES_LOOK_LEFT_L_FB` moves the left eyelid consistent with leftward gaze.



`XR_FACE_EXPRESSION2_EYES_LOOK_LEFT_R_FB` moves the right eyelid consistent with leftward gaze.

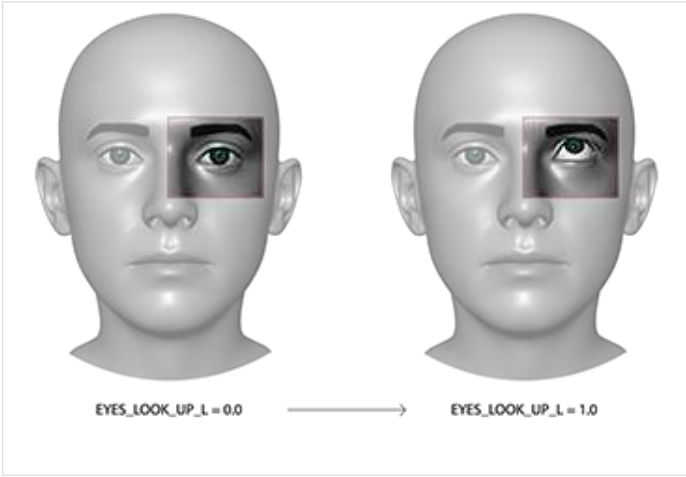


`XR_FACE_EXPRESSION2_EYES_LOOK_RIGHT_L_FB` moves the left eyelid consistent with rightward gaze.

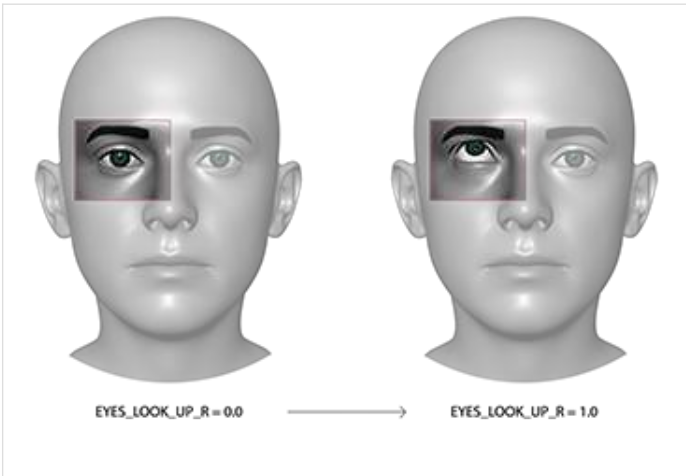


`XR_FACE_EXPRESSION2_EYES_LOOK_RIGHT_R_FB` moves the right eyelid consistent with rightward gaze.

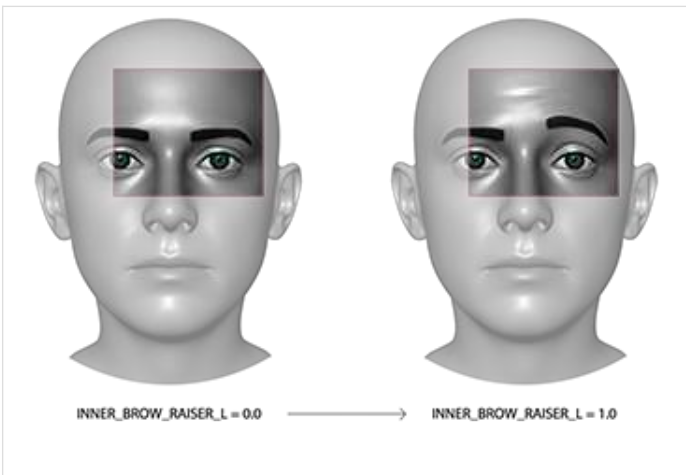




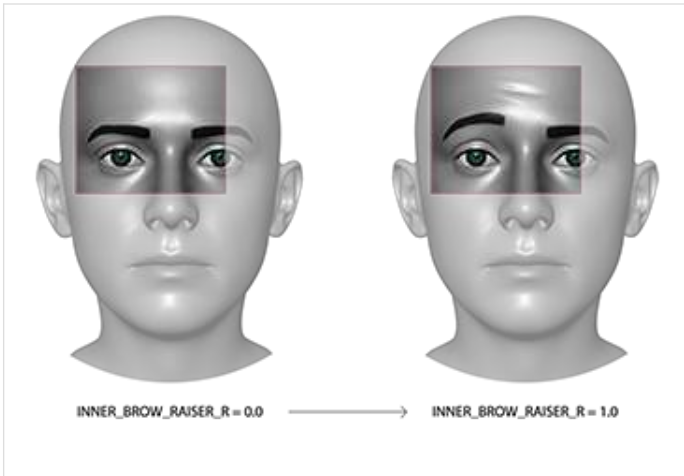
`XR_FACE_EXPRESSION2_EYES_LOOK_UP_L_FB` moves the left eyelid consistent with upward gaze.



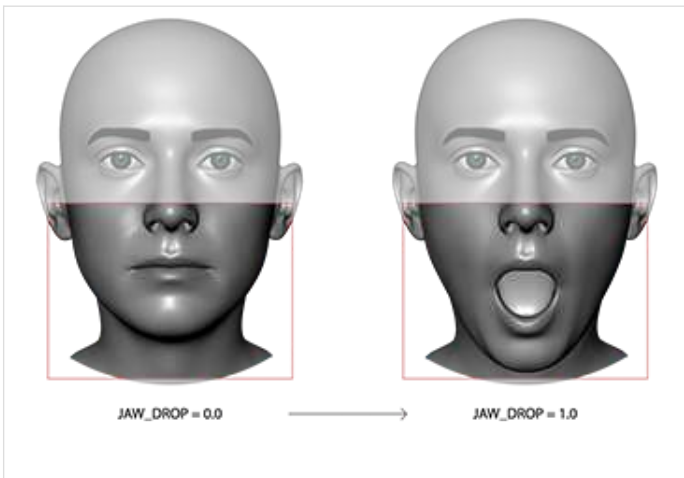
`XR_FACE_EXPRESSION2_EYES_LOOK_UP_R_FB` moves the right eyelid consistent with upward gaze.



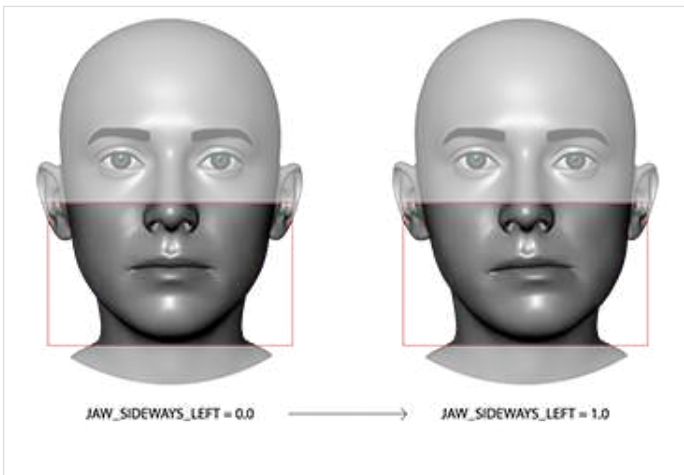
`XR_FACE_EXPRESSION2_INNER_BROW_RAISER_L_FB` lifts the left medial brow and forehead area.



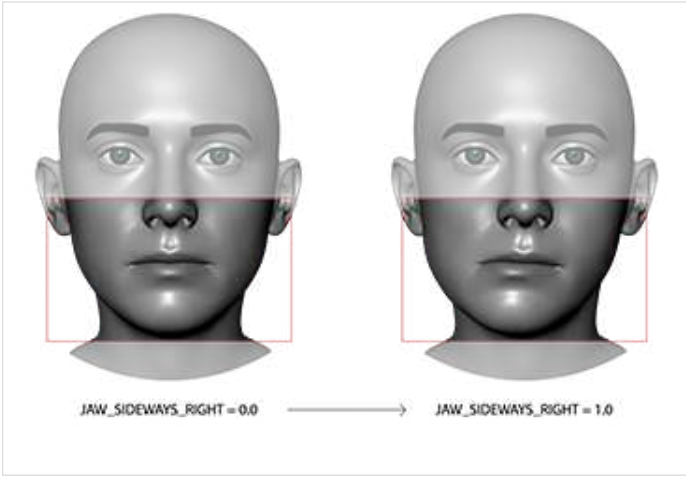
`XR_FACE_EXPRESSION2_INNER_BROW_RAISER_R_FB` lifts the right medial brow and forehead area.



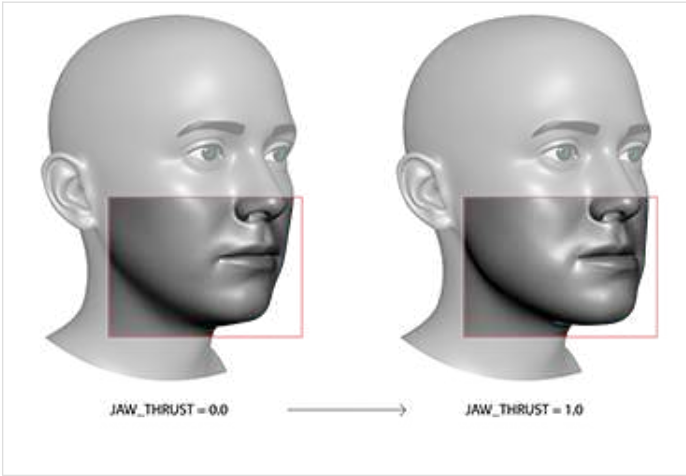
`XR_FACE_EXPRESSION2_JAW_DROP_FB` moves the lower mandible downward and toward the neck.



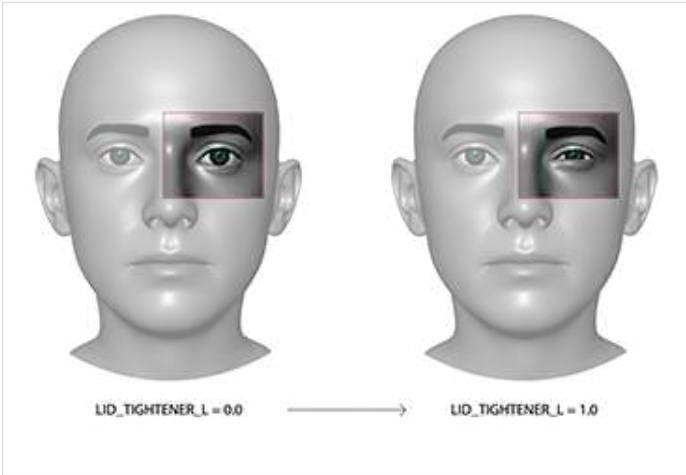
`XR_FACE_EXPRESSION2_JAW_SIDeways_LEFT_FB` moves the lower mandible leftward.



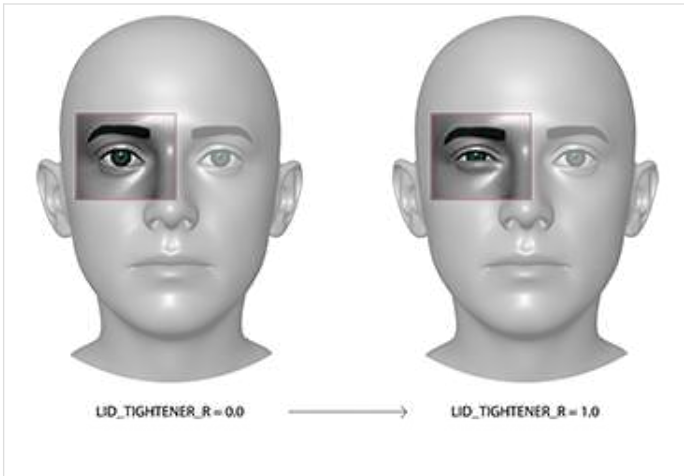
`XR_FACE_EXPRESSION2_JAW_SIDeways_RIGHT_FB` moves the lower mandible rightward.



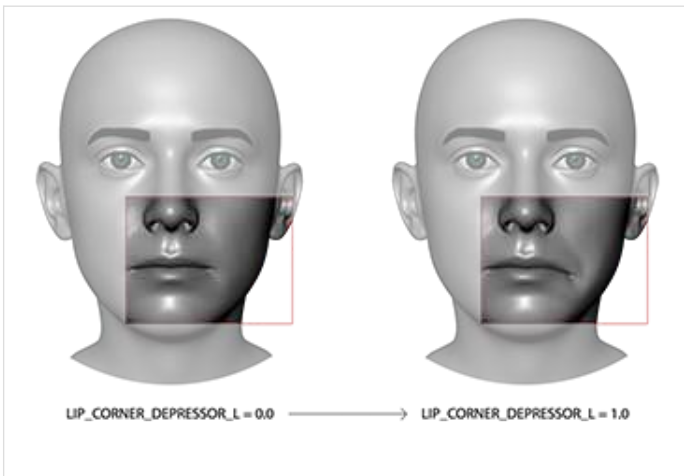
`XR_FACE_EXPRESSION2_JAW_THRUST_FB` projects the lower mandible forward.



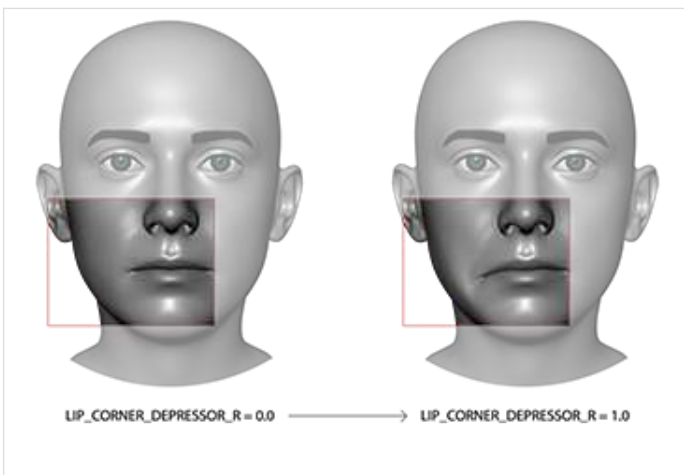
`XR_FACE_EXPRESSION2_LID_TIGHTENER_L_FB` tightens the rings around the left eyelid and pushes the lower eyelid skin toward the inner eye corners.



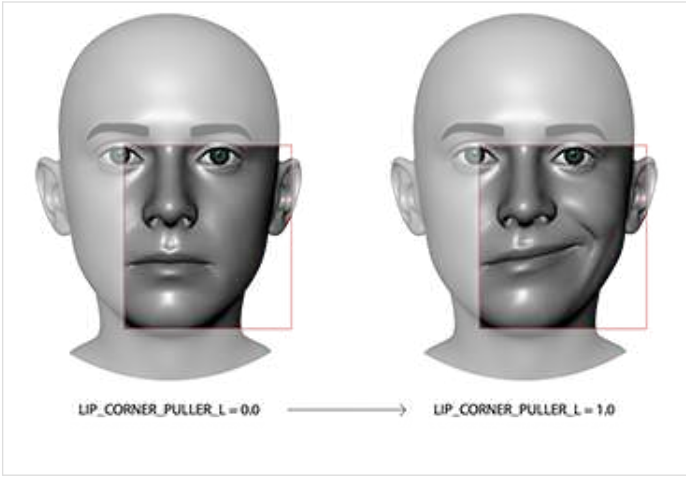
`XR_FACE_EXPRESSION2_LID_TIGHTENER_R_FB` tightens the rings around the right eyelid and pushes the lower eyelid skin toward the inner eye corners.



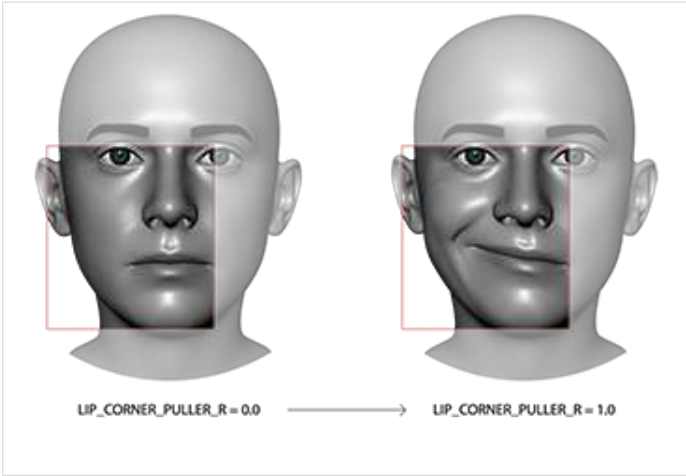
`XR_FACE_EXPRESSION2_LIP_CORNER_DEPRESSOR_L_FB` draws the left lip corner downward.



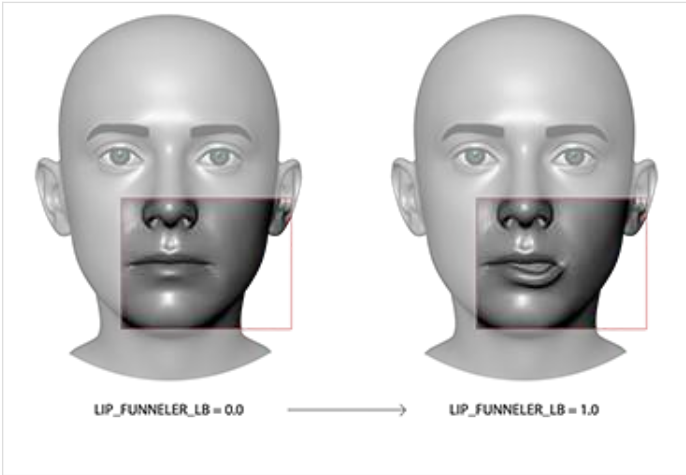
`XR_FACE_EXPRESSION2_LIP_CORNER_DEPRESSOR_R_FB` draws the right lip corner downward.



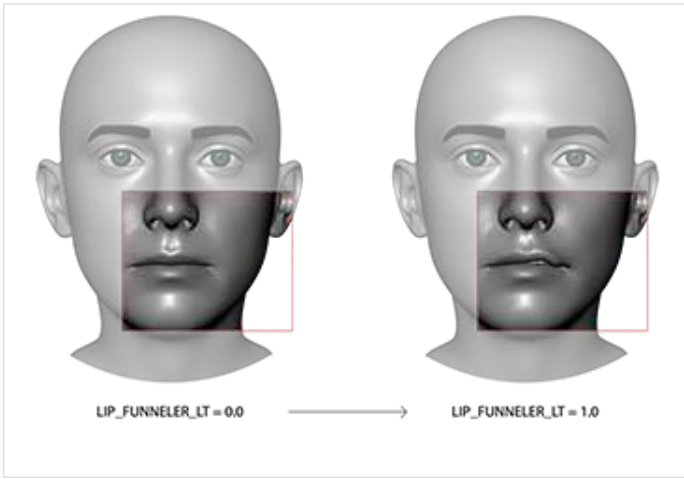
**XR\_FACE\_EXPRESSION2\_LIP\_CORNER\_PULLER\_L\_FB**  
draws the left lip corners up, back, and laterally.



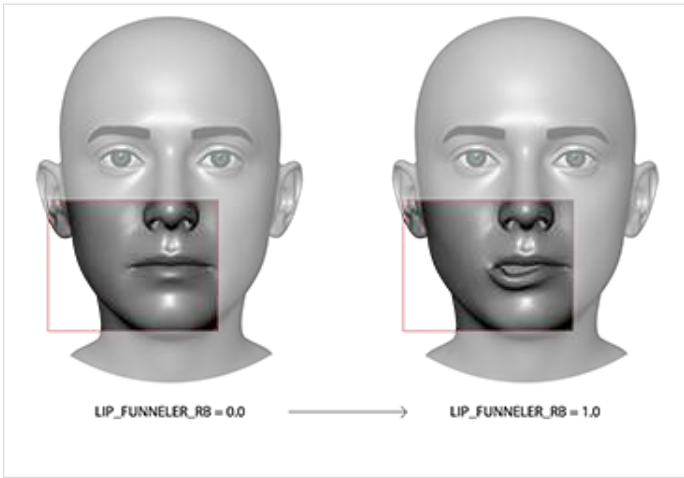
**XR\_FACE\_EXPRESSION2\_LIP\_CORNER\_PULLER\_R\_FB**  
draws the right lip corners up, back, and laterally.



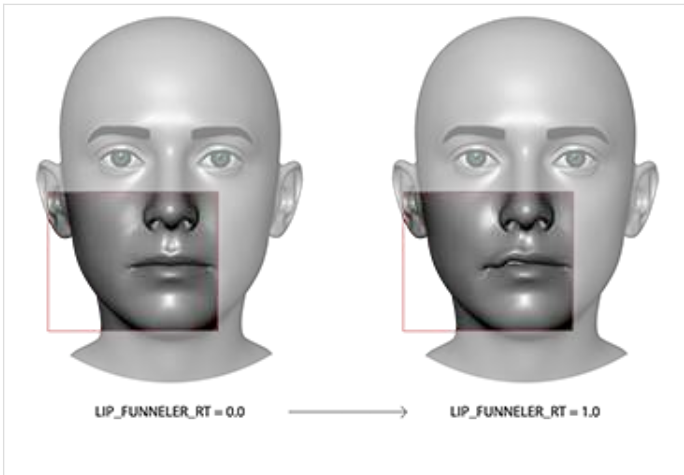
**XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_LB\_FB**  
fans the left bottom lip outward in a forward projection,  
often rounding the mouth and separating the lips.



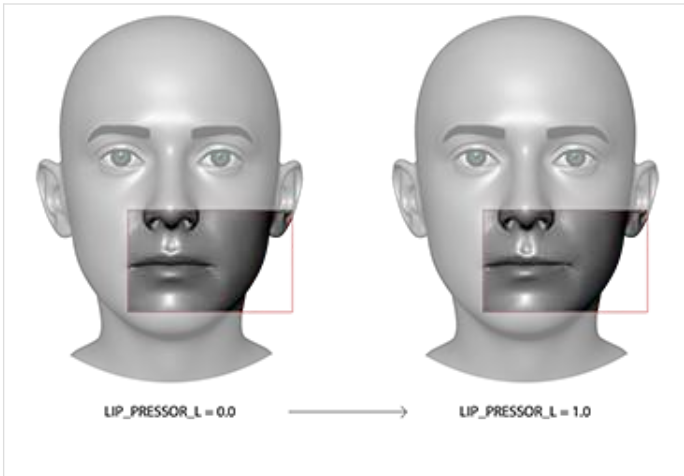
**XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_LT\_FB** fans the left top lip outward in a forward projection, often rounding the mouth and separating the lips.



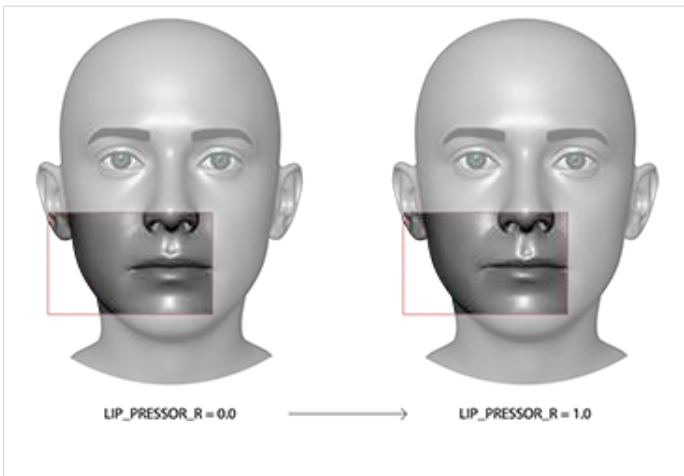
**XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_RB\_FB** fans the right bottom lip outward in a forward projection, often rounding the mouth and separating the lips.



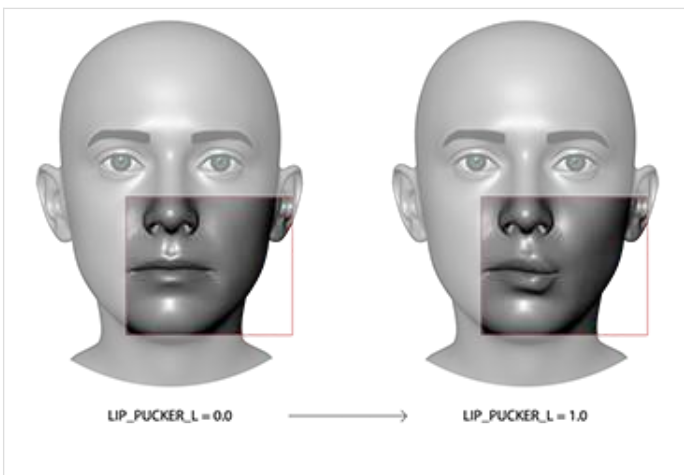
**XR\_FACE\_EXPRESSION2\_LIP\_FUNNELER\_RT\_FB** fans the right top lip outward in a forward projection, often rounding the mouth and separating the lips.



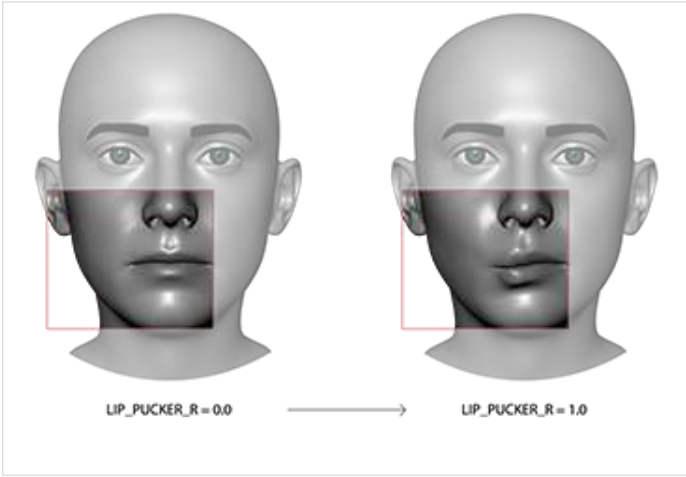
`XR_FACE_EXPRESSION2_LIP_PRESSOR_L_FB` presses the left upper and left lower lips against one another.



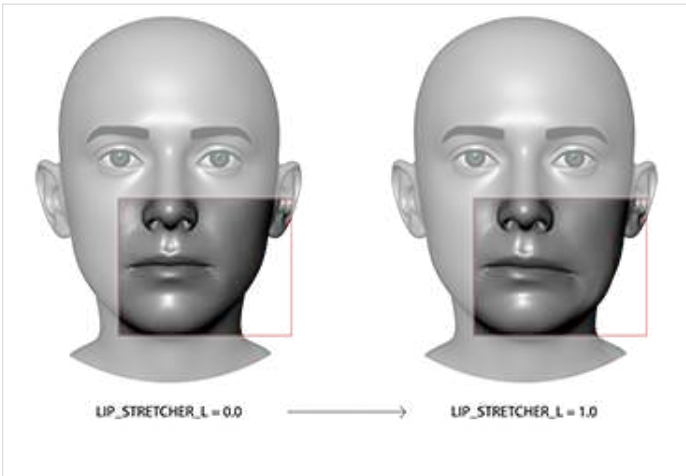
`XR_FACE_EXPRESSION2_LIP_PRESSOR_R_FB` presses the right upper and right lower lips against one another.



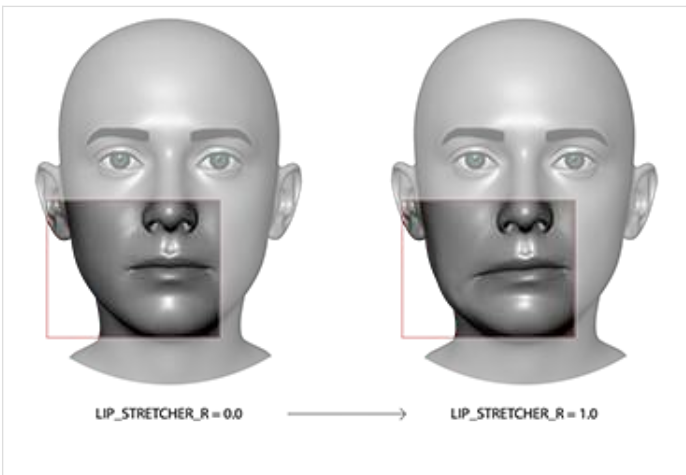
`XR_FACE_EXPRESSION2_LIP_PUCKER_L_FB` draws the left lip corners medially causing the lips protrude in the process.



`XR_FACE_EXPRESSION2_LIP_PUCKER_R_FB` draws the right lip corners medially causing the lips protrude in the process.

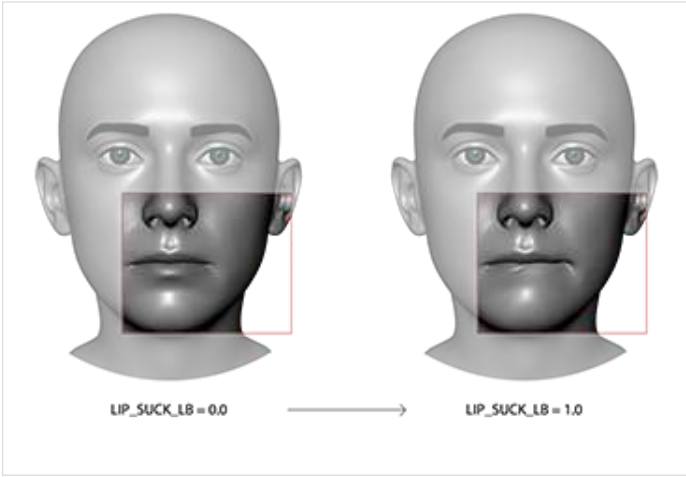


`XR_FACE_EXPRESSION2_LIP_STRETCHER_L_FB` draws the left lip corners laterally, stretching the lips and widening the jawline.

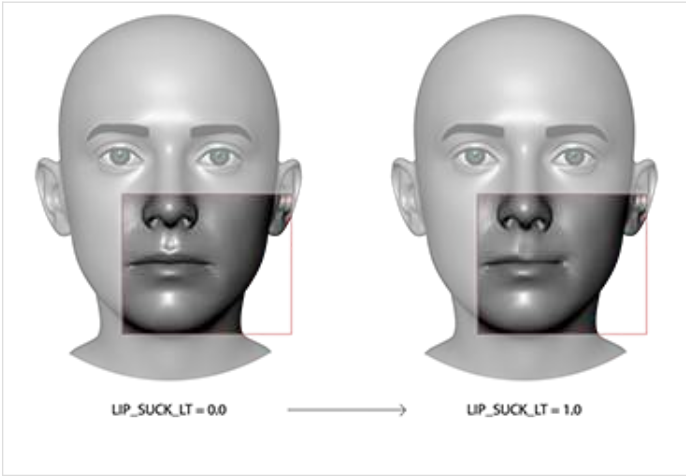


`XR_FACE_EXPRESSION2_LIP_STRETCHER_R_FB` draws the right lip corners laterally, stretching the lips and widening the jawline.

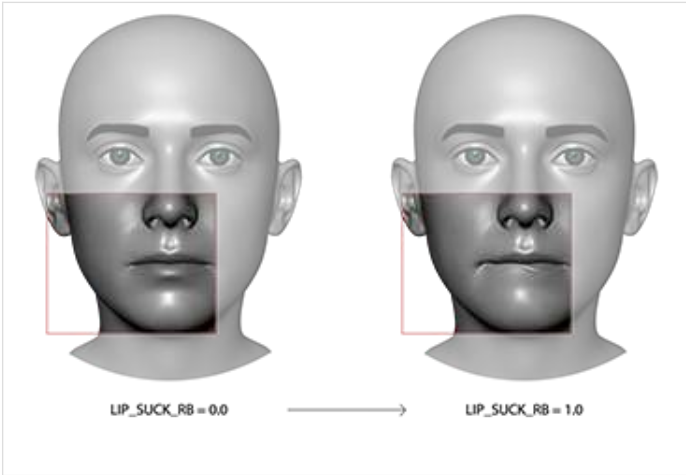




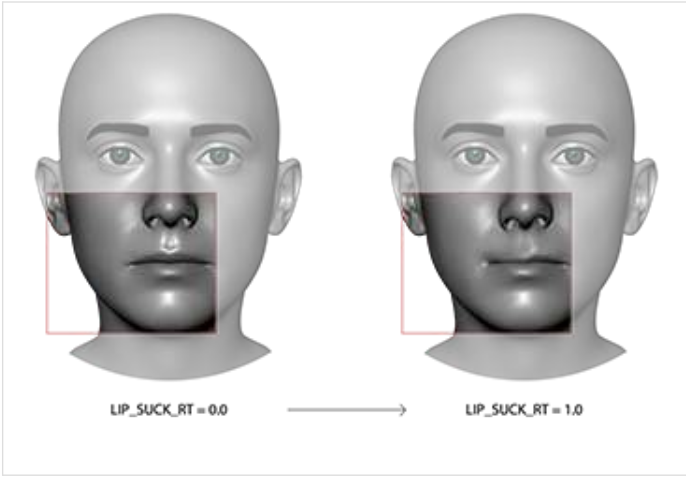
`XR_FACE_EXPRESSION2_LIP_SUCK_LB_FB` sucks the left bottom lip toward the inside of the mouth.



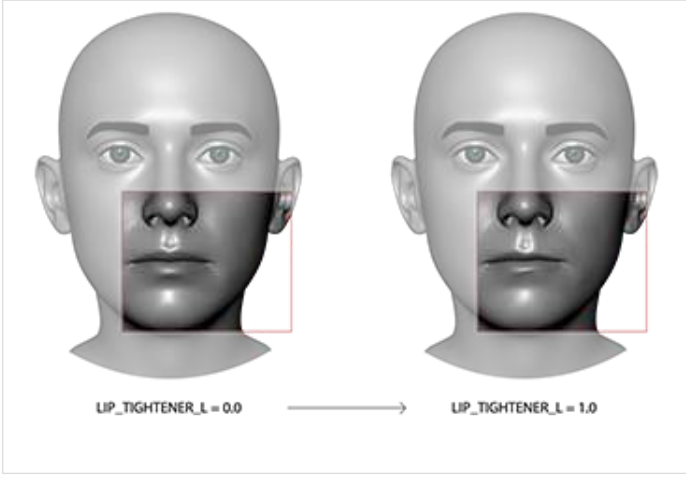
`XR_FACE_EXPRESSION2_LIP_SUCK_LT_FB` sucks the left top lip toward the inside of the mouth.



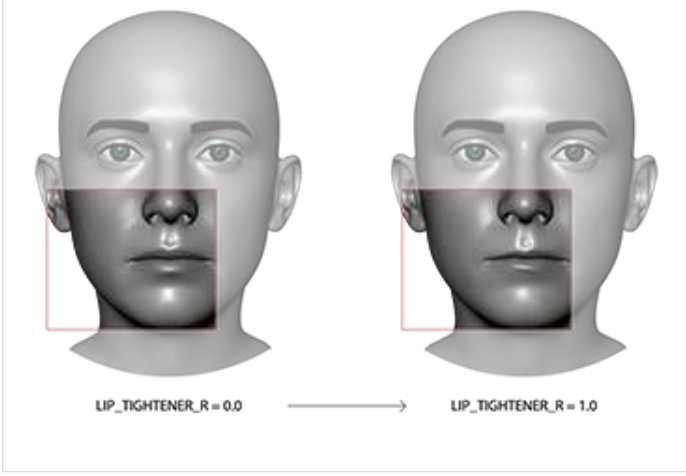
`XR_FACE_EXPRESSION2_LIP_SUCK_RB_FB` sucks the right bottom lip toward the inside of the mouth.



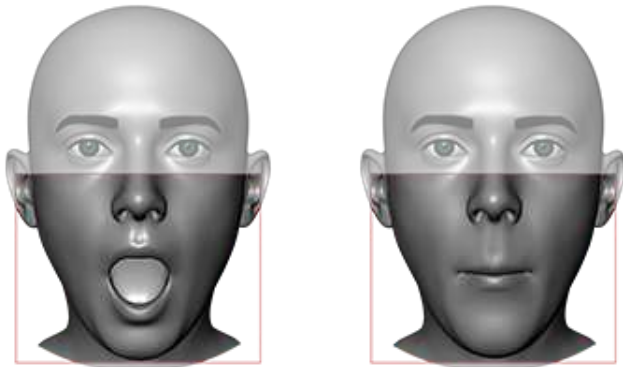
`XR_FACE_EXPRESSION2_LIP_SUCK_RT_FB` sucks the right top lip toward the inside of the mouth.



`XR_FACE_EXPRESSION2_LIP_TIGHTENER_L_FB` narrows or constricts the left lips on a horizontal plane.



`XR_FACE_EXPRESSION2_LIP_TIGHTENER_R_FB` narrows or constricts the right lips on a horizontal plane.

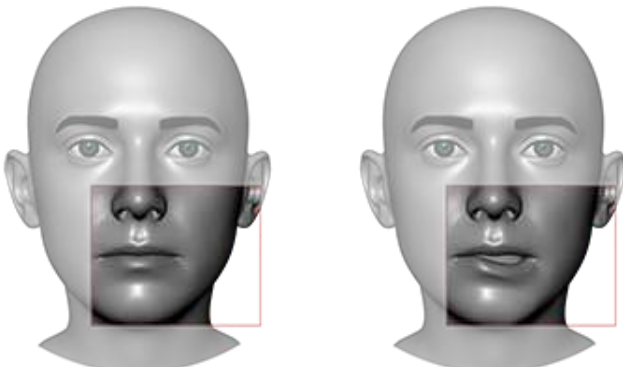


JAW\_DROP = 1.0  
LIPS\_TOWARD = 0.0



JAW\_DROP = 1.0  
LIPS\_TOWARD = 1.0

**XR\_FACE\_EXPRESSION2\_LIPS\_TOWARD\_FB** forces contact between top and bottom lips to keep the mouth closed regardless of the position of the jaw.

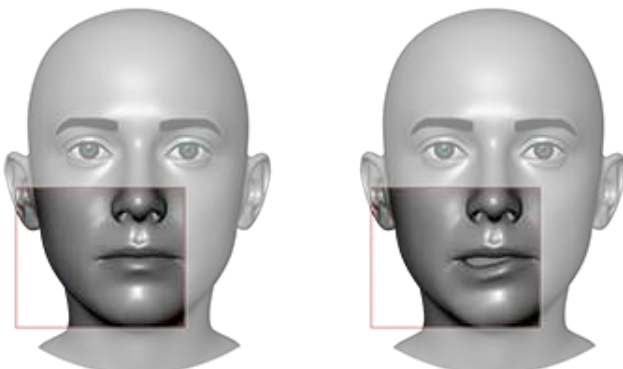


LOWER\_LIP\_DEPRESSOR\_L = 0.0



LOWER\_LIP\_DEPRESSOR\_L = 1.0

**XR\_FACE\_EXPRESSION2\_LOWER\_LIP\_DEPRESSOR\_L\_FB** draws the left lower lip downward and slightly laterally.

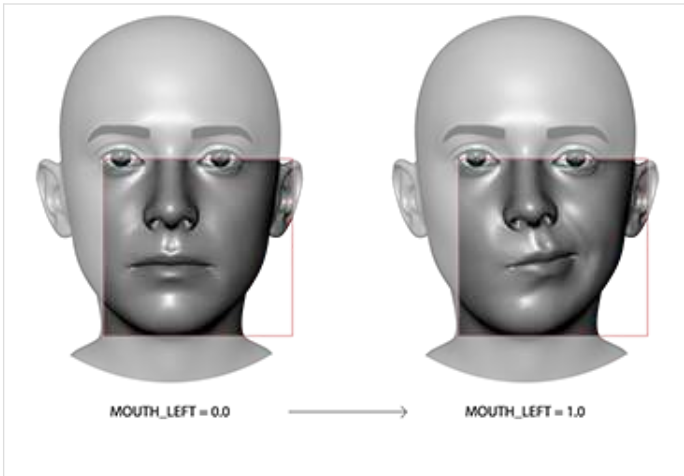


LOWER\_LIP\_DEPRESSOR\_R = 0.0

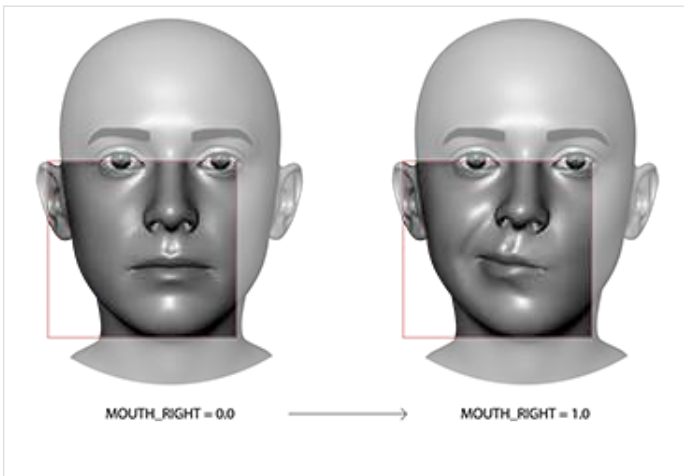


LOWER\_LIP\_DEPRESSOR\_R = 1.0

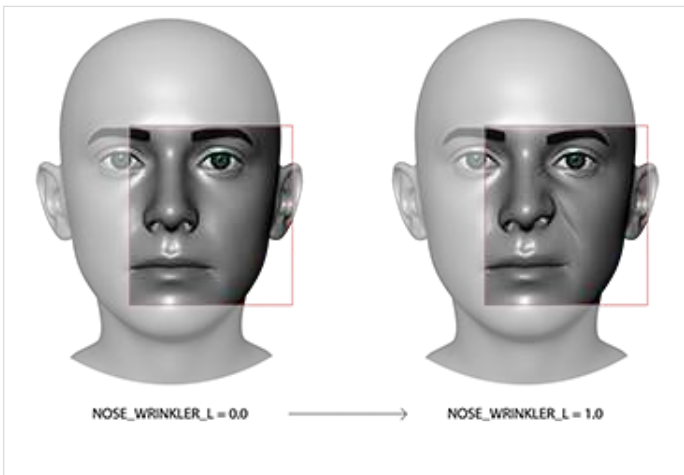
**XR\_FACE\_EXPRESSION2\_LOWER\_LIP\_DEPRESSOR\_R\_FB** draws the right lower lip downward and slightly laterally.



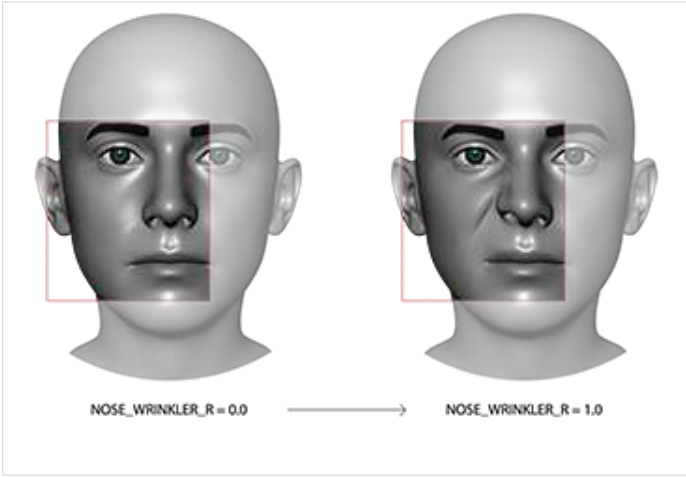
**XR\_FACE\_EXPRESSION2\_MOUTH\_LEFT\_FB** pulls the left lip corner leftward and pushes the right side of the mouth toward the left lip corner.



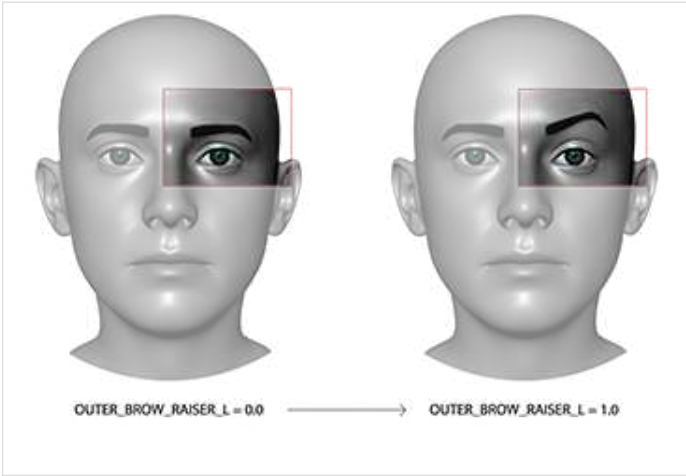
**XR\_FACE\_EXPRESSION2\_MOUTH\_RIGHT\_FB** pulls the right lip corner rightward and pushes the left side of the mouth toward the right lip corner.



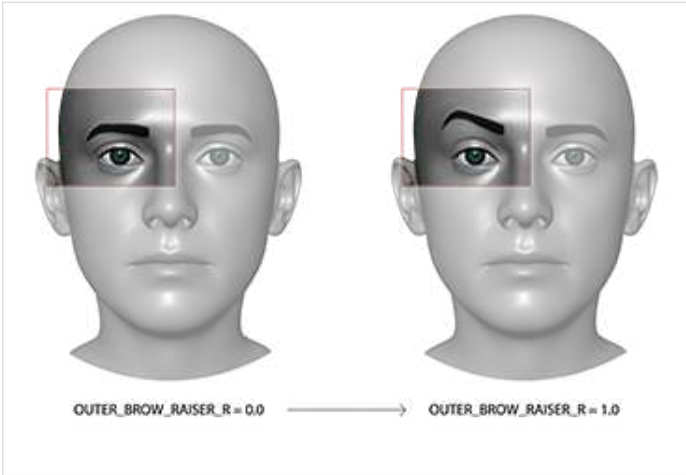
**XR\_FACE\_EXPRESSION2\_NOSE\_WRINKLER\_L\_FB** lifts the left sides of the nose, nostrils, and central upper lip area. Often pairs with brow lowering muscles to lower the medial brow tips.



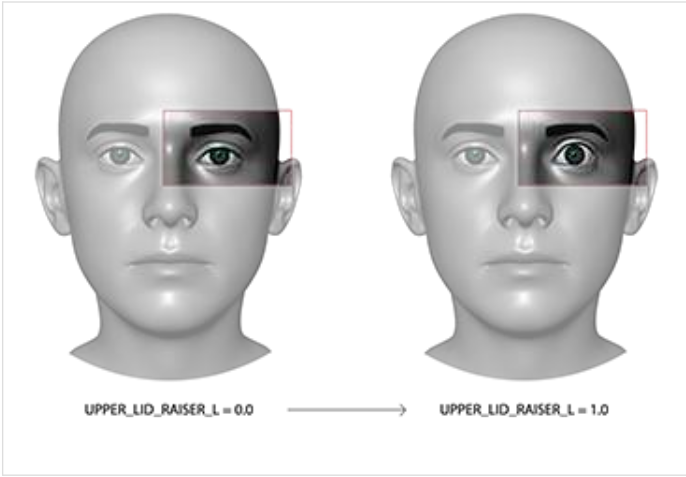
**XR\_FACE\_EXPRESSION2\_NOSE\_WRINKLER\_R\_FB** lifts the right sides of the nose, nostrils, and central upper lip area. Often pairs with brow lowering muscles to lower the medial brow tips.



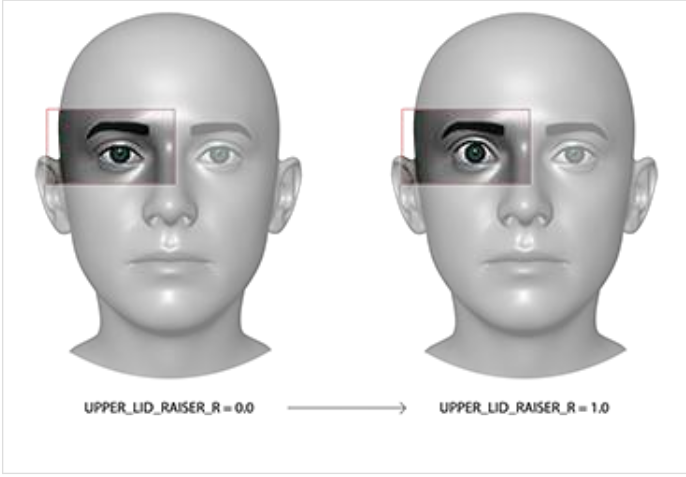
**XR\_FACE\_EXPRESSION2\_OUTER\_BROW\_RAISER\_L\_FB** lifts the lateral left brow and forehead areas.



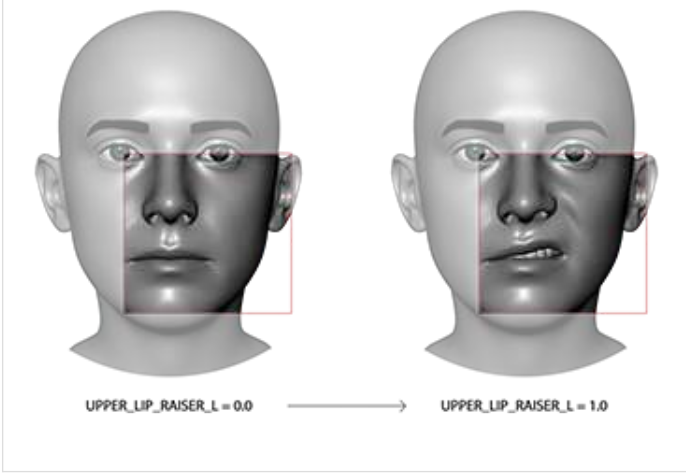
**XR\_FACE\_EXPRESSION2\_OUTER\_BROW\_RAISER\_R\_FB** lifts the lateral right brow and forehead areas.



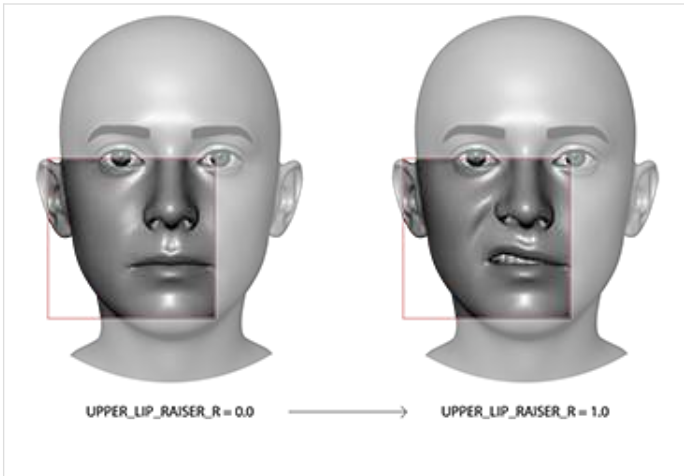
**XR\_FACE\_EXPRESSION2\_UPPER\_LID\_RAISER\_L\_FB** pulls the top left eyelid up and back to widen eyes.



**XR\_FACE\_EXPRESSION2\_UPPER\_LID\_RAISER\_R\_FB** pulls the top right eyelid up and back to widen eyes.



**XR\_FACE\_EXPRESSION2\_UPPER\_LIP\_RAISER\_L\_FB** lifts the top left lip (in a more lateral manner than nose wrinkler).



`XR_FACE_EXPRESSION2_UPPER_LIP_RAISER_R_FB` lifts the top right lip (in a more lateral manner than nose wrinkler).



`XR_FACE_EXPRESSION2_TONGUE_TIP_INTERDENTAL_FB` raises the tip of the tongue to touch the top teeth like with the viseme "TH". The tongue is visible and slightly sticks out past the teeth line.



`XR_FACE_EXPRESSION2_TONGUE_TIP_ALVEOLAR_FB` raises the tip of tongue to touch the back of the top teeth like in the viseme "NN".



`XR_FACE_EXPRESSION2_TONGUE_FRONT_DORSAL_PALATE_FB` makes the front part of the tongue to press against the palate like in the viseme "CH".



`XR_FACE_EXPRESSION2_TONGUE_MID_DORSAL_PALATE_FB` presses the middle of the tongue against the palate like in the viseme "DD".



`XR_FACE_EXPRESSION2_TONGUE_BACK_DORSAL_VELAR_FB` presses the back of the tongue against the palate like in the viseme "KK".



`XR_FACE_EXPRESSION2_TONGUE_OUT_FB` sticks the tongue out.



`XR_FACE_EXPRESSION2_TONGUE_RETREAT_FB` pulls the tongue back in the throat and makes the tongue to stay down like in the viseme "AA".

### 12.53.8. Conventions of confidence areas

This extension defines two separate areas of confidence.



```
// Provided by XR_FB_face_tracking2
typedef enum XrFaceConfidence2FB {
    XR_FACE_CONFIDENCE2_LOWER_FACE_FB = 0,
    XR_FACE_CONFIDENCE2_UPPER_FACE_FB = 1,
    XR_FACE_CONFIDENCE2_COUNT_FB = 2,
    XR_FACE_CONFIDENCE_2FB_MAX_ENUM_FB = 0x7FFFFFFF
} XrFaceConfidence2FB;
```

The "upper face" area represents everything above the upper lip, including the eyes and eyebrows. The "lower face" area represents everything under the eyes, including the mouth and chin. Cheek and nose areas contribute to both "upper face" and "lower face" areas.

## New Object Types

- [XrFaceTracker2FB](#)

## New Flag Types

## New Enum Constants

[XrObjectType](#) enumeration is extended with:

- [XR\\_OBJECT\\_TYPE\\_FACE\\_TRACKER2\\_FB](#)

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SYSTEM\\_FACE\\_TRACKING\\_PROPERTIES2\\_FB](#)
- [XR\\_TYPE\\_FACE\\_TRACKER\\_CREATE\\_INFO2\\_FB](#)
- [XR\\_TYPE\\_FACE\\_EXPRESSION\\_INFO2\\_FB](#)
- [XR\\_TYPE\\_FACE\\_EXPRESSION\\_WEIGHTS2\\_FB](#)

## New Enums

- [XrFaceExpression2FB](#)
- [XrFaceExpressionSet2FB](#)
- [XrFaceConfidence2FB](#)
- [XrFaceTrackingDataSource2FB](#)

## New Structures

- [XrSystemFaceTrackingProperties2FB](#)
- [XrFaceTrackerCreateInfo2FB](#)
- [XrFaceExpressionInfo2FB](#)

- [XrFaceExpressionWeights2FB](#)

## New Functions

- [xrCreateFaceTracker2FB](#)
- [xrDestroyFaceTracker2FB](#)
- [xrGetFaceExpressionWeights2FB](#)

## Issues

- Should we add the tongue shapes to [XR\\_FB\\_face\\_tracking](#) as a new enum value in [XrFaceExpressionSetFB](#)?
  - **Resolved.** We expect that all applications should use [XR\\_FB\\_face\\_tracking2](#) in the future and that [XR\\_FB\\_face\\_tracking](#) will ultimately be replaced by this extension.

## Version History

- Revision 1, 2023-10-06 (Jaebong Lee)
  - Initial extension description

# 12.54. XR\_FB\_foveation

## Name String

[XR\\_FB\\_foveation](#)

## Extension Type

Instance extension

## Registered Extension Number

115

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_swapchain\\_update\\_state](#)

## Contributors

Kevin Xiao, Facebook

Ross Ning, Facebook

Remi Palandri, Facebook

Cass Everitt, Facebook

## Overview

Foveation in the context of XR is a rendering technique that allows the area of an image near the focal point or fovea of the eye to be displayed at higher resolution than areas in the periphery. This trades some visual fidelity in the periphery, where it is less noticeable for the user, for improved rendering performance, most notably regarding the fragment shader, as fewer pixels or subpixels in the periphery need to be shaded and processed. On platforms which support foveation patterns and features tailored towards the optical properties, performance profiles, and hardware support of specific HMDs, application developers may request and use available foveation profiles from the runtime. Foveation profiles refer to a set of properties describing how, when, and where foveation will be applied.

This extension allows:

- An application to create swapchains that can support foveation for its graphics API.
- An application to request foveation profiles supported by the runtime and apply them to foveation-supported swapchains.

In order to enable the functionality of this extension, you **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo` `enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

```
XR_DEFINE_HANDLE(XrFoveationProfileFB)
```

`XrFoveationProfileFB` represents a set of properties and resources that define a foveation pattern for the runtime, which **can** be applied to individual swapchains.

## New Flag Types

```
typedef XrFlags64 XrSwapchainCreateFoveationFlagsFB;
```

```
// Flag bits for XrSwapchainCreateFoveationFlagsFB
static const XrSwapchainCreateFoveationFlagsFB
XR_SWAPCHAIN_CREATE_FOVEATION_SCALED_BIN_BIT_FB = 0x00000001;
static const XrSwapchainCreateFoveationFlagsFB
XR_SWAPCHAIN_CREATE_FOVEATION_FRAGMENT_DENSITY_MAP_BIT_FB = 0x00000002;
```

## Flag Descriptions

- **XR\_SWAPCHAIN\_CREATE\_FOVEATION\_SCALED\_BIN\_BIT\_FB** — Explicitly create the swapchain with scaled bin foveation support. The application must ensure that the swapchain is using the OpenGL graphics API and that the QCOM\_texture\_foveated extension is supported and enabled.
- **XR\_SWAPCHAIN\_CREATE\_FOVEATION\_FRAGMENT\_DENSITY\_MAP\_BIT\_FB** — Explicitly create the swapchain with fragment density map foveation support. The application must ensure that the swapchain is using the Vulkan graphics API and that the VK\_EXT\_fragment\_density\_map extension is supported and enabled.

```
typedef XrFlags64 XrSwapchainStateFoveationFlagsFB;
```

```
// Flag bits for XrSwapchainStateFoveationFlagsFB
```

There are currently no foveation swapchain state flags. This is reserved for future use.

## New Enum Constants

[XrObjectType](#) enumeration is extended with:

- **XR\_OBJECT\_TYPE\_FOVEATION\_PROFILE\_FB**

[XrStructureType](#) enumeration is extended with:

- **XR\_TYPE\_FOVEATION\_PROFILE\_CREATE\_INFO\_FB**
- **XR\_TYPE\_SWAPCHAIN\_CREATE\_INFO\_FOVEATION\_FB**
- **XR\_TYPE\_SWAPCHAIN\_STATE\_FOVEATION\_FB**

## New Enums

## New Structures

[XrFoveationProfileCreateInfoFB](#) **must** be provided when calling [xrCreateFoveationProfileFB](#). The runtime **must** interpret [XrFoveationProfileCreateInfoFB](#) without any additional structs in its [next](#) chain as a request to create a foveation profile that will apply no foveation to any area of the swapchain.

The [XrFoveationProfileCreateInfoFB](#) structure is defined as:

```
// Provided by XR_FB_foveation
typedef struct XrFoveationProfileCreateInfoFB {
    XrStructureType    type;
    void*              next;
} XrFoveationProfileCreateInfoFB;
```

### Member Descriptions

- [type](#) is the [XrStructureType](#) of this structure.
- [next](#) is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

### Valid Usage (Implicit)

- The [XR\\_FB\\_foveation](#) extension **must** be enabled prior to using [XrFoveationProfileCreateInfoFB](#)
- [type](#) **must** be `XR_TYPE_FOVEATION_PROFILE_CREATE_INFO_FB`
- [next](#) **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrFoveationLevelProfileCreateInfoFB](#)

[XrSwapchainCreateInfoFoveationFB](#) **can** be provided in the [next](#) chain of [XrSwapchainCreateInfo](#) when calling [xrCreateSwapchain](#) to indicate to the runtime that the swapchain **must** be created with foveation support in the corresponding graphics API. [XrSwapchainCreateInfoFoveationFB](#) contains additional foveation-specific flags for swapchain creation.

The [XrSwapchainCreateInfoFoveationFB](#) structure is defined as:

```
// Provided by XR_FB_foveation
typedef struct XrSwapchainCreateInfoFoveationFB {
    XrStructureType          type;
    void*                    next;
    XrSwapchainCreateFoveationFlagsFB  flags;
} XrSwapchainCreateInfoFoveationFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of [XrSwapchainCreateFoveationFlagBitsFB](#) which indicate various characteristics for how foveation is enabled on the swapchain.

## Valid Usage (Implicit)

- The [XR\\_FB\\_foveation](#) extension **must** be enabled prior to using [XrSwapchainCreateInfoFoveationFB](#)
- `type` **must** be `XR_TYPE_SWAPCHAIN_CREATE_INFO_FOVEATION_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be `0` or a valid combination of [XrSwapchainCreateFoveationFlagBitsFB](#) values

[XrSwapchainStateFoveationFB](#) **can** be provided in place of [XrSwapchainStateBaseHeaderFB](#) when calling [xrUpdateSwapchainFB](#) to update the foveation properties of the swapchain. [XrSwapchainCreateInfoFoveationFB](#) contains the desired foveation profile and additional foveation specific flags for updating the swapchain.

The [XrSwapchainStateFoveationFB](#) structure is defined as:

```
// Provided by XR_FB_foveation
typedef struct XrSwapchainStateFoveationFB {
    XrStructureType          type;
    void*                    next;
    XrSwapchainStateFoveationFlagsFB  flags;
    XrFoveationProfileFB     profile;
} XrSwapchainStateFoveationFB;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **flags** is a bitmask of [XrSwapchainStateFoveationFlagBitsFB](#) which indicate various characteristics of how and when the foveation properties of the swapchain **must** be updated.
- **profile** is an [XrFoveationProfileFB](#) defining the desired foveation properties to be applied to the swapchain.

## Valid Usage (Implicit)

- The [XR\\_FB\\_foveation](#) extension **must** be enabled prior to using [XrSwapchainStateFoveationFB](#)
- **type** **must** be `XR_TYPE_SWAPCHAIN_STATE_FOVEATION_FB`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **flags** **must** be `0`
- **profile** **must** be a valid [XrFoveationProfileFB](#) handle

## New Functions

The [xrCreateFoveationProfileFB](#) function is defined as:

```
// Provided by XR_FB_foveation
XrResult xrCreateFoveationProfileFB(
    XrSession session,
    const XrFoveationProfileCreateInfoFB* createInfo,
    XrFoveationProfileFB* profile);
```

## Parameter Descriptions

- **session** is the [XrSession](#) that created the swapchains to which this foveation profile will be applied.
- **createInfo** is a pointer to an [XrFoveationProfileCreateInfoFB](#) structure containing parameters to be used to create the foveation profile.
- **profile** is a pointer to a handle in which the created [XrFoveationProfileFB](#) is returned.

Creates an [XrFoveationProfileFB](#) handle. The returned foveation profile handle **may** be subsequently used in API calls.

### Valid Usage (Implicit)

- The [XR\\_FB\\_foveation](#) extension **must** be enabled prior to calling [xrCreateFoveationProfileFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrFoveationProfileCreateInfoFB](#) structure
- `profile` **must** be a pointer to an [XrFoveationProfileFB](#) handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The [xrDestroyFoveationProfileFB](#) function is defined as:

```
// Provided by XR_FB_foveation
XrResult xrDestroyFoveationProfileFB(
    XrFoveationProfileFB          profile);
```



## Parameter Descriptions

- `profile` is the `XrFoveationProfileFB` to destroy.

`XrFoveationProfileFB` handles are destroyed using `xrDestroyFoveationProfileFB`. A `XrFoveationProfileFB` **may** be safely destroyed after being applied to a swapchain state using `xrUpdateSwapchainFB` without affecting the foveation parameters of the swapchain. The application is responsible for ensuring that it has no calls using `profile` in progress when the foveation profile is destroyed.

## Valid Usage (Implicit)

- The `XR_FB_foveation` extension **must** be enabled prior to calling `xrDestroyFoveationProfileFB`
- `profile` **must** be a valid `XrFoveationProfileFB` handle

## Thread Safety

- Access to `profile`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`

## Issues

## Version History

- Revision 1, 2021-05-13 (Kevin Xiao)
  - Initial extension description

# 12.55. XR\_FB\_foveation\_configuration

## Name String

`XR_FB_foveation_configuration`

## Extension Type

Instance extension

## Registered Extension Number

116

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_FB\\_foveation](#)

## Contributors

Kevin Xiao, Facebook  
Ross Ning, Facebook  
Remi Palandri, Facebook  
Cass Everitt, Facebook  
Gloria Kennickell, Facebook

## Overview

On Facebook HMDs, developers **may** create foveation profiles generated by the runtime for the optical properties and performance profile of the specific HMD.

This extension allows:

- An application to request foveation profiles generated by the runtime for the current HMD.

In order to enable the functionality of this extension, you **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo.enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_FOVEATION_LEVEL_PROFILE_CREATE_INFO_FB`

## New Enums

The possible foveation levels are specified by the `XrFoveationLevelFB` enumeration:

```
// Provided by XR_FB_foveation_configuration
typedef enum XrFoveationLevelFB {
    XR_FOVEATION_LEVEL_NONE_FB = 0,
    XR_FOVEATION_LEVEL_LOW_FB = 1,
    XR_FOVEATION_LEVEL_MEDIUM_FB = 2,
    XR_FOVEATION_LEVEL_HIGH_FB = 3,
    XR_FOVEATION_LEVEL_MAX_ENUM_FB = 0x7FFFFFFF
} XrFoveationLevelFB;
```

### Enumerant Descriptions

- `XR_FOVEATION_LEVEL_NONE_FB` — No foveation
- `XR_FOVEATION_LEVEL_LOW_FB` — Less foveation (higher periphery visual fidelity, lower performance)
- `XR_FOVEATION_LEVEL_MEDIUM_FB` — Medium foveation (medium periphery visual fidelity, medium performance)
- `XR_FOVEATION_LEVEL_HIGH_FB` — High foveation (lower periphery visual fidelity, higher performance)

The possible foveation levels are specified by the `XrFoveationDynamicFB` enumeration:

```
// Provided by XR_FB_foveation_configuration
typedef enum XrFoveationDynamicFB {
    XR_FOVEATION_DYNAMIC_DISABLED_FB = 0,
    XR_FOVEATION_DYNAMIC_LEVEL_ENABLED_FB = 1,
    XR_FOVEATION_DYNAMIC_MAX_ENUM_FB = 0x7FFFFFFF
} XrFoveationDynamicFB;
```

## Enumerant Descriptions

- `XR_FOVEATION_DYNAMIC_DISABLED_FB` — Static foveation at the maximum desired level
- `XR_FOVEATION_DYNAMIC_LEVEL_ENABLED_FB` — Dynamic changing foveation based on performance headroom available up to the maximum desired level

## New Structures

`XrFoveationLevelProfileCreateInfoFB` can be provided in the `next` chain of `XrFoveationProfileCreateInfoFB` when calling `xrCreateFoveationProfileFB`. The runtime **must** interpret `XrSwapchainCreateInfoFoveationFB` with `XrFoveationLevelProfileCreateInfoFB` in its `next` chain as a request to create a foveation profile that will apply a fixed foveation pattern according to the parameters defined in the `XrFoveationLevelProfileCreateInfoFB`.

The `XrFoveationLevelProfileCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_foveation_configuration
typedef struct XrFoveationLevelProfileCreateInfoFB {
    XrStructureType    type;
    void*              next;
    XrFoveationLevelFB level;
    float              verticalOffset;
    XrFoveationDynamicFB dynamic;
} XrFoveationLevelProfileCreateInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `level` is the maximum desired foveation level.
- `verticalOffset` is the desired vertical offset in degrees for the center of the foveation pattern.
- `dynamic` is the desired dynamic foveation setting.

## Valid Usage (Implicit)

- The [XR\\_FB\\_foveation\\_configuration](#) extension **must** be enabled prior to using [XrFoveationLevelProfileCreateInfoFB](#)
- **type** **must** be [XR\\_TYPE\\_FOVEATION\\_LEVEL\\_PROFILE\\_CREATE\\_INFO\\_FB](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#). See also: [XrFoveationEyeTrackedProfileCreateInfoMETA](#)
- **level** **must** be a valid [XrFoveationLevelFB](#) value
- **dynamic** **must** be a valid [XrFoveationDynamicFB](#) value

### New Functions

### Issues

### Version History

- Revision 1, 2021-05-13 (Kevin Xiao)
  - Initial extension description

## 12.56. XR\_FB\_foveation\_vulkan

### Name String

[XR\\_FB\\_foveation\\_vulkan](#)

### Extension Type

Instance extension

### Registered Extension Number

161

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_foveation](#)

### Contributors

Kevin Xiao, Facebook

Ross Ning, Facebook

Remi Palandri, Facebook

Cass Everitt, Facebook  
Gloria Kennickell, Facebook

## Overview

The Vulkan graphics API requires an image to be applied to the swapchain to apply a foveation pattern.

This extension allows:

- An application to obtain foveation textures or constructs needed for foveated rendering in Vulkan.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo](#) `enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SWAPCHAIN_IMAGE_FOVEATION_VULKAN_FB`

## New Enums

## New Structures

[XrSwapchainImageFoveationVulkanFB](#) **can** be provided in the `next` chain of [XrSwapchainImageVulkanKHR](#) when calling [xrEnumerateSwapchainImages](#) on a swapchain created with [xrCreateSwapchain](#), if [XrSwapchainCreateInfoFoveationFB](#) was in the `next` chain of [XrSwapchainCreateInfo](#) and [XrSwapchainCreateInfoFoveationFB](#) had the `XR_SWAPCHAIN_CREATE_FOVEATION_FRAGMENT_DENSITY_MAP_BIT_FB` flag set. The `image`, `width`, and `height` will be populated by [xrEnumerateSwapchainImages](#) to be compatible with the corresponding [XrSwapchainImageVulkanKHR](#).

The [XrSwapchainImageFoveationVulkanFB](#) structure is defined as:

```
// Provided by XR_FB_foveation_vulkan
typedef struct XrSwapchainImageFoveationVulkanFB {
    XrStructureType    type;
    void*              next;
    VkImage             image;
    uint32_t           width;
    uint32_t           height;
} XrSwapchainImageFoveationVulkanFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `image` is a valid Vulkan `VkImage` to use.
- `width` is the horizontal width in pixels of the image.
- `height` is the vertical height in pixels of the image.

## Valid Usage (Implicit)

- The [XR\\_FB\\_foveation\\_vulkan](#) extension **must** be enabled prior to using [XrSwapchainImageFoveationVulkanFB](#)
- `type` **must** be `XR_TYPE_SWAPCHAIN_IMAGE_FOVEATION_VULKAN_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

## Issues

## Version History

- Revision 1, 2021-05-26 (Kevin Xiao)
  - Initial extension description

# 12.57. XR\_FB\_hand\_tracking\_aim

## Name String

`XR_FB_hand_tracking_aim`

## Extension Type

Instance extension

## Registered Extension Number

112

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_EXT\\_hand\\_tracking](#)

## Contributors

Federico Schliemann, Facebook

James Hillery, Facebook

Gloria Kennickell, Facebook

## Overview

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.

This extension allows:

- An application to get a set of basic gesture states for the hand when using the [XR\\_EXT\\_hand\\_tracking](#) extension.

## New Object Types

## New Flag Types

```
typedef XrFlags64 XrHandTrackingAimFlagsFB;
```



```

// Flag bits for XrHandTrackingAimFlagsFB
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_COMPUTED_BIT_FB = 0x00000001;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_VALID_BIT_FB = 0x00000002;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_INDEX_PINCHING_BIT_FB =
0x00000004;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_MIDDLE_PINCHING_BIT_FB =
0x00000008;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_RING_PINCHING_BIT_FB =
0x00000010;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_LITTLE_PINCHING_BIT_FB =
0x00000020;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_SYSTEM_GESTURE_BIT_FB =
0x00000040;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_DOMINANT_HAND_BIT_FB =
0x00000080;
static const XrHandTrackingAimFlagsFB XR_HAND_TRACKING_AIM_MENU_PRESSED_BIT_FB =
0x00000100;

```

## Flag Descriptions

- `XR_HAND_TRACKING_AIM_COMPUTED_BIT_FB` — Aiming data is computed from additional sources beyond the hand data in the base structure
- `XR_HAND_TRACKING_AIM_VALID_BIT_FB` — Aiming data is valid
- `XR_HAND_TRACKING_AIM_INDEX_PINCHING_BIT_FB` — Index finger pinch discrete signal
- `XR_HAND_TRACKING_AIM_MIDDLE_PINCHING_BIT_FB` — Middle finger pinch discrete signal
- `XR_HAND_TRACKING_AIM_RING_PINCHING_BIT_FB` — Ring finger pinch discrete signal
- `XR_HAND_TRACKING_AIM_LITTLE_PINCHING_BIT_FB` — Little finger pinch discrete signal
- `XR_HAND_TRACKING_AIM_SYSTEM_GESTURE_BIT_FB` — System gesture is active
- `XR_HAND_TRACKING_AIM_DOMINANT_HAND_BIT_FB` — Hand is currently marked as dominant for the system
- `XR_HAND_TRACKING_AIM_MENU_PRESSED_BIT_FB` — System menu gesture is active

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_HAND_TRACKING_AIM_STATE_FB`

## New Enums

## New Structures

[XrHandTrackingAimStateFB](#) can be provided in the `next` chain of [XrHandJointLocationsEXT](#) when calling [xrLocateHandJointsEXT](#) to request aiming gesture information associated with this hand.

The [XrHandTrackingAimStateFB](#) structure is defined as:

```
// Provided by XR_FB_hand_tracking_aim
typedef struct XrHandTrackingAimStateFB {
    XrStructureType      type;
    void*                next;
    XrHandTrackingAimFlagsFB status;
    XrPosef              aimPose;
    float                pinchStrengthIndex;
    float                pinchStrengthMiddle;
    float                pinchStrengthRing;
    float                pinchStrengthLittle;
} XrHandTrackingAimStateFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `status` is a bitmask of [XrHandTrackingAimFlagBitsFB](#) describing the availability and state of other signals.
- `aimPose` is a system-determined "aim" pose, similar in intent and convention to the [aim poses](#) used with the action system, based on hand data.
- `pinchStrengthIndex` is the current pinching strength for the index finger of this hand. Range is 0.0 to 1.0, with 1.0 meaning index and thumb are fully touching.
- `pinchStrengthMiddle` is the current pinching strength for the middle finger of this hand. Range is 0.0 to 1.0, with 1.0 meaning middle and thumb are fully touching.
- `pinchStrengthRing` is the current pinching strength for the ring finger of this hand. Range is 0.0 to 1.0, with 1.0 meaning ring and thumb are fully touching.
- `pinchStrengthLittle` is the current pinching strength for the little finger of this hand. Range is 0.0 to 1.0, with 1.0 meaning little and thumb are fully touching.

## Valid Usage (Implicit)

- The `XR_FB_hand_tracking_aim` extension **must** be enabled prior to using `XrHandTrackingAimStateFB`
- `type` **must** be `XR_TYPE_HAND_TRACKING_AIM_STATE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### New Functions

### Issues

### Version History

- Revision 1, 2021-07-07 (Federico Schliemann)
  - Initial extension description
- Revision 2, 2022-04-20 (John Kearney)
  - Correct next chain parent for `XrHandTrackingAimStateFB` to `XrHandJointLocationsEXT`

## 12.58. XR\_FB\_hand\_tracking\_capsules

### Name String

`XR_FB_hand_tracking_capsules`

### Extension Type

Instance extension

### Registered Extension Number

113

### Revision

3

### Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_EXT\\_hand\\_tracking](#)

### Contributors

Federico Schliemann, Facebook

James Hillery, Facebook

Gloria Kennickell, Facebook

## Overview

The [XR\\_EXT\\_hand\\_tracking](#) extension provides a list of hand joint poses which include a collision sphere for each joint. However some physics systems prefer to use capsules as a collision stand in for the hands.

This extension allows:

- An application to get a list of capsules that represent the volume of the hand when using the [XR\\_EXT\\_hand\\_tracking](#) extension.

## New Object Types

## New Flag Types

## New Enum Constants

- [XR\\_HAND\\_TRACKING\\_CAPSULE\\_POINT\\_COUNT\\_FB](#)
  - [XR\\_FB\\_HAND\\_TRACKING\\_CAPSULE\\_POINT\\_COUNT](#) was the original name, and is still provided as an alias for backward compatibility.
- [XR\\_HAND\\_TRACKING\\_CAPSULE\\_COUNT\\_FB](#)
  - [XR\\_FB\\_HAND\\_TRACKING\\_CAPSULE\\_COUNT](#) was the original name, and is still provided as an alias for backward compatibility.

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_HAND\\_TRACKING\\_CAPSULES\\_STATE\\_FB](#)

## New Enums

## New Structures

The [XrHandCapsuleFB](#) structure is defined as:

```
// Provided by XR_FB_hand_tracking_capsules
typedef struct XrHandCapsuleFB {
    XrVector3f      points[XR_HAND_TRACKING_CAPSULE_POINT_COUNT_FB];
    float          radius;
    XrHandJointEXT joint;
} XrHandCapsuleFB;
```

It describes a collision capsule associated with a hand joint.

## Member Descriptions

- `points` are the two points defining the capsule length.
- `radius` is the radius of the capsule.
- `joint` is the hand joint that drives this capsule's transform. Multiple capsules **may** be attached to the same joint.

## Valid Usage (Implicit)

- The `XR_FB_hand_tracking_capsules` extension **must** be enabled prior to using `XrHandCapsuleFB`

`XrHandTrackingCapsulesStateFB` **can** be provided in the `next` chain of `XrHandJointLocationsEXT` when calling `xrLocateHandJointsEXT` to request collision capsule information associated with this hand.

The `XrHandTrackingCapsulesStateFB` structure is defined as:

```
// Provided by XR_FB_hand_tracking_capsules
typedef struct XrHandTrackingCapsulesStateFB {
    XrStructureType    type;
    void*              next;
    XrHandCapsuleFB    capsules[XR_HAND_TRACKING_CAPSULE_COUNT_FB];
} XrHandTrackingCapsulesStateFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `capsules` is an array of capsules.

## Valid Usage (Implicit)

- The `XR_FB_hand_tracking_capsules` extension **must** be enabled prior to using `XrHandTrackingCapsulesStateFB`
- `type` **must** be `XR_TYPE_HAND_TRACKING_CAPSULES_STATE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

## Issues

## Version History

- Revision 1, 2021-07-07 (Federico Schliemann)
  - Initial extension description
- Revision 2, 2021-11-18 (Rylie Pavlik, Collabora, Ltd.)
  - Fix typos/naming convention errors: rename `XR_FB_HAND_TRACKING_CAPSULE_POINT_COUNT` to `XR_HAND_TRACKING_CAPSULE_POINT_COUNT_FB` and `XR_FB_HAND_TRACKING_CAPSULE_COUNT` to `XR_HAND_TRACKING_CAPSULE_COUNT_FB`, providing the old names as compatibility aliases.
- Revision 3, 2022-04-20 (John Kearney)
  - Correct next chain parent for `XrHandTrackingCapsulesStateFB` to `XrHandJointLocationsEXT`

# 12.59. XR\_FB\_hand\_tracking\_mesh

## Name String

`XR_FB_hand_tracking_mesh`

## Extension Type

Instance extension

## Registered Extension Number

111

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_EXT\\_hand\\_tracking](#)

## Contributors

Federico Schliemann, Facebook  
James Hillery, Facebook  
Gloria Kennickell, Facebook

## Overview

The [XR\\_EXT\\_hand\\_tracking](#) extension provides a list of hand joint poses but no mechanism to render a skinned hand mesh.

This extension allows:

- An application to get a skinned hand mesh and a bind pose skeleton that **can** be used to render a hand object driven by the joints from the [XR\\_EXT\\_hand\\_tracking](#) extension.
- Control the scale of the hand joints returned by [XR\\_EXT\\_hand\\_tracking](#).

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_HAND\\_TRACKING\\_MESH\\_FB](#)
- [XR\\_TYPE\\_HAND\\_TRACKING\\_SCALE\\_FB](#)

## New Enums

## New Structures

The [XrVector4sFB](#) structure is defined as:

```
// Provided by XR_FB_hand_tracking_mesh
typedef struct XrVector4sFB {
    int16_t    x;
    int16_t    y;
    int16_t    z;
    int16_t    w;
} XrVector4sFB;
```

This is a short integer, four component vector type, used for per-vertex joint indexing for mesh skinning.

## Member Descriptions

- **x** is the **x** component of the vector.
- **y** is the **y** component of the vector.
- **z** is the **z** component of the vector.
- **w** is the **w** component of the vector.

## Valid Usage (Implicit)

- The `XR_FB_hand_tracking_mesh` extension **must** be enabled prior to using `XrVector4sFB`

The `XrHandTrackingMeshFB` structure contains three sets of parallel, application-allocated arrays: one with per-joint data, one with vertex data, and one with index data.

The `XrHandTrackingMeshFB` structure is defined as:

```
// Provided by XR_FB_hand_tracking_mesh
typedef struct XrHandTrackingMeshFB {
    XrStructureType    type;
    void*              next;
    uint32_t           jointCapacityInput;
    uint32_t           jointCountOutput;
    XrPosef*           jointBindPoses;
    float*             jointRadii;
    XrHandJointEXT*    jointParents;
    uint32_t           vertexCapacityInput;
    uint32_t           vertexCountOutput;
    XrVector3f*        vertexPositions;
    XrVector3f*        vertexNormals;
    XrVector2f*        vertexUVs;
    XrVector4sFB*     vertexBlendIndices;
    XrVector4f*        vertexBlendWeights;
    uint32_t           indexCapacityInput;
    uint32_t           indexCountOutput;
    int16_t*           indices;
} XrHandTrackingMeshFB;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `jointCapacityInput` is the capacity of the joint data arrays in this structure, or `0` to indicate a request to retrieve the required capacity.
- `jointCountOutput` is filled in by the runtime with the count of joint data elements written, or the required capacity in the case that any of `jointCapacityInput`, `vertexCapacityInput`, or `indexCapacityInput` is insufficient.
- `jointBindPoses` is an array of poses that matches what is returned by [xrLocateHandJointsEXT](#) which describes the hand skeleton's bind pose.
- `jointRadii` is an array of joint radii at bind pose.
- `jointParents` is an array of joint parents to define a bone hierarchy for the hand skeleton.
- `vertexCapacityInput` is the capacity of the vertex data arrays in this structure, or `0` to indicate a request to retrieve the required capacity.
- `vertexCountOutput` is filled in by the runtime with the count of vertex data elements written, or the required capacity in the case that any of `jointCapacityInput`, `vertexCapacityInput`, or `indexCapacityInput` is insufficient.
- `vertexPositions` is an array of 3D vertex positions.
- `vertexNormals` is an array of 3D vertex normals.
- `vertexUVs` is an array of texture coordinates for this vertex.
- `vertexBlendIndices` is an array of bone blend indices.
- `vertexBlendWeights` is an array of bone blend weights.
- `indexCapacityInput` is the capacity of the index data arrays in this structure, or `0` to indicate a request to retrieve the required capacity.
- `indexCountOutput` is filled in by the runtime with the count of index data elements written, or the required capacity in the case that any of `jointCapacityInput`, `vertexCapacityInput`, or `indexCapacityInput` is insufficient.
- `indices` is an array of triangle indices.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the array sizes in the "struct form" as used here.

All arrays are application-allocated, and all **may** be `NULL` if any of `jointCapacityInput`, `vertexCapacityInput`, or `indexCapacityInput` is `0`.

The data in a fully-populated [XrHandTrackingMeshFB](#) is immutable during the lifetime of the

corresponding [XrInstance](#), and is intended to be retrieved once then used in combination with data changing per-frame retrieved from [xrLocateHandJointsEXT](#).

### Valid Usage (Implicit)

- The [XR\\_FB\\_hand\\_tracking\\_mesh](#) extension **must** be enabled prior to using [XrHandTrackingMeshFB](#)
- `type` **must** be `XR_TYPE_HAND_TRACKING_MESH_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `jointCapacityInput` is not `0`, `jointBindPoses` **must** be a pointer to an array of `jointCapacityInput` [XrPosef](#) structures
- If `jointCapacityInput` is not `0`, `jointRadii` **must** be a pointer to an array of `jointCapacityInput` `float` values
- If `jointCapacityInput` is not `0`, `jointParents` **must** be a pointer to an array of `jointCapacityInput` [XrHandJointEXT](#) values
- If `vertexCapacityInput` is not `0`, `vertexPositions` **must** be a pointer to an array of `vertexCapacityInput` [XrVector3f](#) structures
- If `vertexCapacityInput` is not `0`, `vertexNormals` **must** be a pointer to an array of `vertexCapacityInput` [XrVector3f](#) structures
- If `vertexCapacityInput` is not `0`, `vertexUVs` **must** be a pointer to an array of `vertexCapacityInput` [XrVector2f](#) structures
- If `vertexCapacityInput` is not `0`, `vertexBlendIndices` **must** be a pointer to an array of `vertexCapacityInput` [XrVector4sFB](#) structures
- If `vertexCapacityInput` is not `0`, `vertexBlendWeights` **must** be a pointer to an array of `vertexCapacityInput` [XrVector4f](#) structures
- If `indexCapacityInput` is not `0`, `indices` **must** be a pointer to an array of `indexCapacityInput` `int16_t` values

[XrHandTrackingScaleFB](#) **can** be provided in the `next` chain of [XrHandJointLocationsEXT](#) when calling [xrLocateHandJointsEXT](#) to indicate to the runtime that the requested joints need to be scaled to a different size and to query the existing scale value. This is useful in breaking up the overall scale out of the skinning transforms.

The [XrHandTrackingScaleFB](#) structure is defined as:

```
// Provided by XR_FB_hand_tracking_mesh
typedef struct XrHandTrackingScaleFB {
    XrStructureType    type;
    void*              next;
    float               sensorOutput;
    float               currentOutput;
    XrBool32            overrideHandScale;
    float               overrideValueInput;
} XrHandTrackingScaleFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `sensorOutput` is an output value: the currently measured scale as otherwise applied without passing this structure.
- `currentOutput` is an output value: the effective output that the bind skeleton is getting on the current call, which **may** be subject to filtering, scaling, or validation.
- `overrideHandScale` indicates whether the runtime **must** scale the output of this [xrLocateHandJointsEXT](#) call according to `overrideValueInput`
- `overrideValueInput` is an **optional** input value, enabled only when the `overrideHandScale` parameter is set. Setting this to 1.0 and setting `overrideHandScale` to `true` will give the joints in mesh binding scale.

## Valid Usage (Implicit)

- The [XR\\_FB\\_hand\\_tracking\\_mesh](#) extension **must** be enabled prior to using [XrHandTrackingScaleFB](#)
- `type` **must** be `XR_TYPE_HAND_TRACKING_SCALE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrGetHandMeshFB](#) function is defined as:

```
// Provided by XR_FB_hand_tracking_mesh
XrResult xrGetHandMeshFB(
    XrHandTrackerEXT          handTracker,
    XrHandTrackingMeshFB*    mesh);
```

## Parameter Descriptions

- **handTracker** is the [XrHandTrackerEXT](#) that is associated with a particular hand.
- **mesh** is the [XrHandTrackingMeshFB](#) output structure.

The [xrGetHandMeshFB](#) function populates an [XrHandTrackingMeshFB](#) structure with enough information to render a skinned mesh driven by the hand joints. As discussed in the specification for that structure, the data enumerated by this call is constant during the lifetime of an [XrInstance](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_hand\\_tracking\\_mesh](#) extension **must** be enabled prior to calling [xrGetHandMeshFB](#)
- **handTracker** **must** be a valid [XrHandTrackerEXT](#) handle
- **mesh** **must** be a pointer to an [XrHandTrackingMeshFB](#) structure

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_SIZE\\_INSUFFICIENT](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

## Issues

### Version History

- Revision 1, 2021-07-07 (Federico Schliemann)
  - Initial extension description
- Revision 2, 2022-04-20 (John Kearney)
  - Correct next chain parent for `XrHandTrackingScaleFB` to `XrHandJointLocationsEXT`
- Revision 3, 2022-07-07 (Rylie Pavlik, Collabora, Ltd.)
  - Correct markup and thus generated valid usage for two-call idiom.

## 12.60. XR\_FB\_haptic\_amplitude\_envelope

### Name String

`XR_FB_haptic_amplitude_envelope`

### Extension Type

Instance extension

### Registered Extension Number

174

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-06-27

### IP Status

No known IP claims.

### Contributors

Aanchal Dalmia, Meta

Federico Schliemann, Meta

### 12.60.1. Overview

This extension enables applications to trigger haptic effect using an Amplitude Envelope buffer.

#### Trigger haptics

An application can trigger an amplitude envelope haptic effect by creating a [XrHapticAmplitudeEnvelopeVibrationFB](#) structure and calling [xrApplyHapticFeedback](#).

The [XrHapticAmplitudeEnvelopeVibrationFB](#) structure is defined as:

```
// Provided by XR_FB_haptic_amplitude_envelope
typedef struct XrHapticAmplitudeEnvelopeVibrationFB {
    XrStructureType    type;
    const void*        next;
    XrDuration          duration;
    uint32_t            amplitudeCount;
    const float*        amplitudes;
} XrHapticAmplitudeEnvelopeVibrationFB;
```

This structure describes an amplitude envelope haptic effect.

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `duration` is the duration of the haptic effect in nanoseconds. See [Duration](#) for more details.
- `amplitudeCount` is the number of samples in the buffer.
- `amplitudes` is the pointer to a float array that contains the samples.

The runtime **should** resample the provided samples in the `amplitudes`, and maintain an internal buffer which **should** be of [XR\\_MAX\\_HAPTIC\\_AMPLITUDE\\_ENVELOPE\\_SAMPLES\\_FB](#) length. The resampling **should** happen based on the `duration`, `amplitudeCount`, and the device's sample rate.

### Valid Usage (Implicit)

- The [XR\\_FB\\_haptic\\_amplitude\\_envelope](#) extension **must** be enabled prior to using [XrHapticAmplitudeEnvelopeVibrationFB](#)
- `type` **must** be `XR_TYPE_HAPTIC_AMPLITUDE_ENVELOPE_VIBRATION_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `amplitudes` **must** be a pointer to an array of `amplitudeCount` float values
- The `amplitudeCount` parameter **must** be greater than 0

## New Object Types

## New Flag Types

## New Enum Constants

- `XR_TYPE_HAPTIC_AMPLITUDE_ENVELOPE_VIBRATION_FB`

## New Defines

```
// Provided by XR_FB_haptic_amplitude_envelope
#define XR_MAX_HAPTIC_AMPLITUDE_ENVELOPE_SAMPLES_FB 4000u
```

`XR_MAX_HAPTIC_AMPLITUDE_ENVELOPE_SAMPLES_FB` defines the maximum number of sample the runtime **should** store in memory.

## New Enums

## New Structures

- `XrHapticAmplitudeEnvelopeVibrationFB`

## New Functions

## Issues

## Version History

- Revision 1, 2022-06-27 (Aanchal Dalmia)
  - Initial extension description

# 12.61. XR\_FB\_haptic\_pcm

## Name String

`XR_FB_haptic_pcm`

## Extension Type

Instance extension

## Registered Extension Number

210

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-06-27

## IP Status

No known IP claims.

## Contributors

Aanchal Dalmia, Meta

Adam Bengis, Meta

### 12.61.1. Overview

This extension enables applications to trigger haptic effects using Pulse Code Modulation (PCM) buffers.

#### Trigger haptics

An application **can** trigger PCM haptic effect by creating a [XrHapticPcmVibrationFB](#) structure and calling [xrApplyHapticFeedback](#).

The [XrHapticPcmVibrationFB](#) structure is defined as:

```
// Provided by XR_FB_haptic_pcm
typedef struct XrHapticPcmVibrationFB {
    XrStructureType    type;
    const void*        next;
    uint32_t           bufferSize;
    const float*       buffer;
    float              sampleRate;
    XrBool32           append;
    uint32_t*          samplesConsumed;
} XrHapticPcmVibrationFB;
```



## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `bufferSize` is the number of samples in the buffer.
- `buffer` is a pointer to a float array representing the PCM samples. If you consider the haptic effect as a sampled analog audio, then this buffer will contain the samples representing that effect. The values in this buffer are expected to be in the range [-1.0, 1.0].
- `sampleRate` is the number of samples to be played per second, this is used to determine the duration of the haptic effect.
- `append` if set to `XR_FALSE`, any existing samples will be cleared and a new haptic effect will begin, if `XR_TRUE`, samples will be appended to the currently playing effect
- `samplesConsumed` is a pointer to an unsigned integer; it is populated by runtime, to tell the application about how many samples were consumed from the input `buffer`

This structure describes a PCM haptic effect.

The runtime **may** resample the provided samples in the `buffer`, and maintain an internal buffer which **should** be of `XR_MAX_HAPTIC_PCM_BUFFER_SIZE_FB` length. The resampling **should** happen based on the `sampleRate` and the device's sample rate.

If `append` is `XR_TRUE` and a preceding `XrHapticPcmVibrationFB` haptic effect on this action has not yet completed, then the runtime **must** finish playing the preceding samples and then play the new haptic effect. If a preceding haptic event on this action has not yet completed, and either the preceding effect is not an `XrHapticPcmVibrationFB` haptic effect or `append` is `XR_FALSE`, the runtime **must** cancel the preceding incomplete effects on that action and start playing the new haptic effect, as usual for the core specification.

When `append` is true and a preceding `XrHapticPcmVibrationFB` haptic effect on this action has not yet completed, then the application can provide a different `sampleRate` in the new haptic effect.

The runtime **must** populate the `samplesConsumed` with the count of the samples from `buffer` which were consumed. The `samplesConsumed` is populated before the `xrApplyHapticFeedback` returns.

## Valid Usage (Implicit)

- The `XR_FB_haptic_pcm` extension **must** be enabled prior to using `XrHapticPcmVibrationFB`
- `type` **must** be `XR_TYPE_HAPTIC_PCM_VIBRATION_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `buffer` **must** be a pointer to an array of `bufferSize` `float` values
- `samplesConsumed` **must** be a pointer to a `uint32_t` value
- The `bufferSize` parameter **must** be greater than `0`

## Get the device sample rate

An application **can** use the `xrGetDeviceSampleRateFB` function to get the sample rate of the currently bound device on which the haptic action is triggered. If the application does not want any resampling to occur, then it can use this function to know the currently bound device sample rate, and pass that value in `sampleRate` of `XrHapticPcmVibrationFB`.

```
// Provided by XR_FB_haptic_pcm
XrResult xrGetDeviceSampleRateFB(
    XrSession session,
    const XrHapticActionInfo* hapticActionInfo,
    XrDevicePcmSampleRateGetInfoFB* deviceSampleRate);
```

## Parameter Descriptions

- `session` is the specified `XrSession`.
- `hapticActionInfo` is the `XrHapticActionInfo` used to provide action and subaction paths
- `deviceSampleRate` is a pointer to `XrDevicePcmSampleRateStateFB` which is populated by the runtime.

The runtime **must** use the `hapticActionInfo` to get the sample rate of the currently bound device on which haptics is triggered and populate the `deviceSampleRate` structure. The device is determined by the `XrHapticActionInfo::action` and `XrHapticActionInfo::subactionPath`. If the `hapticActionInfo` is bound to more than one device, then runtime **should** assume that the all these bound devices have the same `deviceSampleRate` and the runtime **should** return the `sampleRate` for any of those bound devices. If the device is invalid, the runtime **must** populate the `deviceSampleRate` of `XrDevicePcmSampleRateStateFB` as `0`. A device can be invalid if the runtime does not find any device (which can play haptics) connected to the headset, or if the device does not support PCM haptic effect.

## Valid Usage (Implicit)

- The `XR_FB_haptic_pcm` extension **must** be enabled prior to calling `xrGetDeviceSampleRateFB`
- `session` **must** be a valid `XrSession` handle
- `hapticActionInfo` **must** be a pointer to a valid `XrHapticActionInfo` structure
- `deviceSampleRate` **must** be a pointer to an `XrDevicePcmSampleRateGetInfoFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_ACTION_TYPE_MISMATCH`
- `XR_ERROR_ACTIONSET_NOT_ATTACHED`

The `XrDevicePcmSampleRateStateFB` structure is defined as:

```
// Provided by XR_FB_haptic_pcm
typedef struct XrDevicePcmSampleRateStateFB {
    XrStructureType    type;
    void*              next;
    float              sampleRate;
} XrDevicePcmSampleRateStateFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `sampleRate` is the sample rate of the currently bound device which can play a haptic effect

## Valid Usage (Implicit)

- The [XR\\_FB\\_haptic\\_pcm](#) extension **must** be enabled prior to using [XrDevicePcmSampleRateStateFB](#)
- `type` **must** be `XR_TYPE_DEVICE_PCM_SAMPLE_RATE_STATE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_HAPTIC_PCM_VIBRATION_FB`
- `XR_TYPE_DEVICE_PCM_SAMPLE_RATE_STATE_FB`

## New Defines

```
// Provided by XR_FB_haptic_pcm
#define XR_MAX_HAPTIC_PCM_BUFFER_SIZE_FB 4000
```

[XR\\_MAX\\_HAPTIC\\_PCM\\_BUFFER\\_SIZE\\_FB](#) defines the maximum number of samples the runtime can store.

## New Enums

## New Structures

- [XrHapticPcmVibrationFB](#)
- [XrDevicePcmSampleRateStateFB](#)

## New Functions

- [xrGetDeviceSampleRateFB](#)

## Issues

## Version History

- Revision 1, 2022-06-27 (Aanchal Dalmia)
  - Initial extension description

# 12.62. XR\_FB\_keyboard\_tracking

## Name String

`XR_FB_keyboard_tracking`

## Extension Type

Instance extension

## Registered Extension Number

117

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Federico Schliemann, Facebook  
Robert Memmott, Facebook  
Cass Everitt, Facebook

## Overview

This extension allows the application to query the system for a supported trackable keyboard type and obtain an [XrSpace](#) handle to track it. It also provides relevant metadata about the keyboard itself, including bounds and a human readable identifier.

## New Object Types

## New Flag Types

```
typedef XrFlags64 XrKeyboardTrackingFlagsFB;
```

```
// Flag bits for XrKeyboardTrackingFlagsFB
static const XrKeyboardTrackingFlagsFB XR_KEYBOARD_TRACKING_EXISTS_BIT_FB = 0x00000001;
static const XrKeyboardTrackingFlagsFB XR_KEYBOARD_TRACKING_LOCAL_BIT_FB = 0x00000002;
static const XrKeyboardTrackingFlagsFB XR_KEYBOARD_TRACKING_REMOTE_BIT_FB = 0x00000004;
static const XrKeyboardTrackingFlagsFB XR_KEYBOARD_TRACKING_CONNECTED_BIT_FB =
0x00000008;
```

## Flag Descriptions

- **XR\_KEYBOARD\_TRACKING\_EXISTS\_BIT\_FB** — indicates that the system has a physically tracked keyboard to report. If not set then no other bits should be considered to be valid or meaningful. If set either **XR\_KEYBOARD\_TRACKING\_LOCAL\_BIT\_FB** or **XR\_KEYBOARD\_TRACKING\_REMOTE\_BIT\_FB** must also be set.
- **XR\_KEYBOARD\_TRACKING\_LOCAL\_BIT\_FB** — indicates that the physically tracked keyboard is intended to be used in a local pairing with the system. Mutually exclusive with **XR\_KEYBOARD\_TRACKING\_REMOTE\_BIT\_FB**.
- **XR\_KEYBOARD\_TRACKING\_REMOTE\_BIT\_FB** — indicates that the physically tracked keyboard is intended to be used while paired to a separate remote computing device. Mutually exclusive with **XR\_KEYBOARD\_TRACKING\_LOCAL\_BIT\_FB**.
- **XR\_KEYBOARD\_TRACKING\_CONNECTED\_BIT\_FB** — indicates that the physically tracked keyboard is actively connected to the headset and capable of sending key data

```
typedef XrFlags64 XrKeyboardTrackingQueryFlagsFB;
```

```
// Flag bits for XrKeyboardTrackingQueryFlagsFB
static const XrKeyboardTrackingQueryFlagsFB XR_KEYBOARD_TRACKING_QUERY_LOCAL_BIT_FB =
0x00000002;
static const XrKeyboardTrackingQueryFlagsFB XR_KEYBOARD_TRACKING_QUERY_REMOTE_BIT_FB =
0x00000004;
```

## Flag Descriptions

- `XR_KEYBOARD_TRACKING_QUERY_LOCAL_BIT_FB` — indicates the query is for the physically tracked keyboard that is intended to be used in a local pairing with the System. Mutually exclusive with `XR_KEYBOARD_TRACKING_QUERY_REMOTE_BIT_FB`.
- `XR_KEYBOARD_TRACKING_QUERY_REMOTE_BIT_FB` — indicates the query is for the physically tracked keyboard that may be connected to a separate remote computing device. Mutually exclusive with `XR_KEYBOARD_TRACKING_QUERY_LOCAL_BIT_FB`.

### New Enum Constants

- `XR_MAX_KEYBOARD_TRACKING_NAME_SIZE_FB`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_KEYBOARD_SPACE_CREATE_INFO_FB`
- `XR_TYPE_KEYBOARD_TRACKING_QUERY_FB`
- `XR_TYPE_SYSTEM_KEYBOARD_TRACKING_PROPERTIES_FB`

### New Enums

### New Structures

The `XrSystemKeyboardTrackingPropertiesFB` structure is defined as:

```
// Provided by XR_FB_keyboard_tracking
typedef struct XrSystemKeyboardTrackingPropertiesFB {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsKeyboardTracking;
} XrSystemKeyboardTrackingPropertiesFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `supportsKeyboardTracking` defines whether the system supports the tracked keyboard feature.

`XrSystemKeyboardTrackingPropertiesFB` is populated with information from the system about tracked

keyboard support.

### Valid Usage (Implicit)

- The `XR_FB_keyboard_tracking` extension **must** be enabled prior to using `XrSystemKeyboardTrackingPropertiesFB`
- `type` **must** be `XR_TYPE_SYSTEM_KEYBOARD_TRACKING_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrKeyboardTrackingQueryFB` structure is defined as:

```
// Provided by XR_FB_keyboard_tracking
typedef struct XrKeyboardTrackingQueryFB {
    XrStructureType          type;
    void*                    next;
    XrKeyboardTrackingQueryFlagsFB flags;
} XrKeyboardTrackingQueryFB;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `flags` is a bitmask of [XrKeyboardTrackingQueryFlagsFB](#).

`XrKeyboardTrackingQueryFB` specifies input data needed to determine which type of tracked keyboard to query for.

### Valid Usage (Implicit)

- The `XR_FB_keyboard_tracking` extension **must** be enabled prior to using `XrKeyboardTrackingQueryFB`
- `type` **must** be `XR_TYPE_KEYBOARD_TRACKING_QUERY_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be a valid combination of [XrKeyboardTrackingQueryFlagBitsFB](#) values
- `flags` **must** not be `0`



The [XrKeyboardTrackingDescriptionFB](#) structure is defined as:

```
// Provided by XR_FB_keyboard_tracking
typedef struct XrKeyboardTrackingDescriptionFB {
    uint64_t          trackedKeyboardId;
    XrVector3f        size;
    XrKeyboardTrackingFlagsFB flags;
    char              name[XR_MAX_KEYBOARD_TRACKING_NAME_SIZE_FB];
} XrKeyboardTrackingDescriptionFB;
```

### Member Descriptions

- `trackedKeyboardId` abstract identifier describing the type of keyboard.
- `size` bounding box.
- `flags` additional information on the type of keyboard available. If `XR_KEYBOARD_TRACKING_EXISTS_BIT_FB` is not set there is no keyboard.
- `name` human readable keyboard identifier.

[XrKeyboardTrackingDescriptionFB](#) describes a trackable keyboard and its associated metadata.

### Valid Usage (Implicit)

- The [XR\\_FB\\_keyboard\\_tracking](#) extension **must** be enabled prior to using [XrKeyboardTrackingDescriptionFB](#)

The [XrKeyboardSpaceCreateInfoFB](#) structure is defined as:

```
// Provided by XR_FB_keyboard_tracking
typedef struct XrKeyboardSpaceCreateInfoFB {
    XrStructureType type;
    void*          next;
    uint64_t       trackedKeyboardId;
} XrKeyboardSpaceCreateInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `trackedKeyboardId` abstract identifier describing the type of keyboard to track.

[XrKeyboardSpaceCreateInfoFB](#) describes a request for the system needed to create a trackable [XrSpace](#) associated with the keyboard.

## Valid Usage (Implicit)

- The [XR\\_FB\\_keyboard\\_tracking](#) extension **must** be enabled prior to using [XrKeyboardSpaceCreateInfoFB](#)
- `type` **must** be `XR_TYPE_KEYBOARD_SPACE_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrQuerySystemTrackedKeyboardFB](#) function is defined as:

```
// Provided by XR_FB_keyboard_tracking
XrResult xrQuerySystemTrackedKeyboardFB(
    XrSession session,
    const XrKeyboardTrackingQueryFB* queryInfo,
    XrKeyboardTrackingDescriptionFB* keyboard);
```

## Parameter Descriptions

- `session` is the session that will be associated with a keyboard space.
- `queryInfo` is the [XrKeyboardTrackingQueryFB](#) that describes the type of keyboard to return. `queryInfo` must have either `XR_KEYBOARD_TRACKING_QUERY_LOCAL_BIT_FB` or `XR_KEYBOARD_TRACKING_QUERY_REMOTE_BIT_FB` set.
- `keyboard` is the [XrKeyboardTrackingDescriptionFB](#) output structure.

The [xrQuerySystemTrackedKeyboardFB](#) function populates an [XrKeyboardTrackingDescriptionFB](#) structure with enough information to describe a keyboard that the system can locate.

## Valid Usage (Implicit)

- The `XR_FB_keyboard_tracking` extension **must** be enabled prior to calling `xrQuerySystemTrackedKeyboardFB`
- `session` **must** be a valid `XrSession` handle
- `queryInfo` **must** be a pointer to a valid `XrKeyboardTrackingQueryFB` structure
- `keyboard` **must** be a pointer to an `XrKeyboardTrackingDescriptionFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrCreateKeyboardSpaceFB` function is defined as:

```
// Provided by XR_FB_keyboard_tracking
XrResult xrCreateKeyboardSpaceFB(
    XrSession session,
    const XrKeyboardSpaceCreateInfoFB* createInfo,
    XrSpace* keyboardSpace);
```

## Parameter Descriptions

- `session` is the session that will be associated with the returned keyboard space.
- `createInfo` is the `XrKeyboardSpaceCreateInfoFB` that describes the type of keyboard to track.
- `keyboardSpace` is the `XrSpace` output structure.

The `xrCreateKeyboardSpaceFB` function returns an `XrSpace` that can be used to locate a physical keyboard in space. The origin of the created `XrSpace` is located in the center of the bounding box in the x and z axes, and at the top of the y axis (meaning the keyboard is located entirely in negative y).

## Valid Usage (Implicit)

- The `XR_FB_keyboard_tracking` extension **must** be enabled prior to calling `xrCreateKeyboardSpaceFB`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrKeyboardSpaceCreateInfoFB` structure
- `keyboardSpace` **must** be a pointer to an `XrSpace` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

## Version History

- Revision 1, 2021-08-27 (Federico Schliemann)
  - Initial extension description

# 12.63. XR\_FB\_passthrough

## Name String

`XR_FB_passthrough`

## Extension Type

Instance extension

## Registered Extension Number

119

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Anton Vaneev, Facebook  
Cass Everitt, Facebook  
Federico Schliemann, Facebook  
Johannes Schmid, Facebook

## Overview

Passthrough is a way to show a user their physical environment in a light-blocking VR headset. Applications may use passthrough in a multitude of ways, including:

- Creating AR-like experiences, where virtual objects augment the user’s environment.
- Bringing real objects into a VR experience.
- Mapping the playspace such that a VR experience is customized to it.

This extension allows:

- An application to request passthrough to be composited with the application content.
- An application to specify the compositing and blending rules between passthrough and VR content.
- An application to apply styles, such as color mapping and edge rendering, to passthrough.
- An application to provide a geometry to be used in place of the user’s physical environment.

Camera images will be projected onto the surface provided by the application. In some cases where a part of the environment, such as a desk, can be approximated well, this provides better visual experience.

## New Object Types

```
XR_DEFINE_HANDLE(XrPassthroughFB)
```

[XrPassthroughFB](#) represents a passthrough feature.

```
XR_DEFINE_HANDLE(XrPassthroughLayerFB)
```

[XrPassthroughLayerFB](#) represents a layer of passthrough content.

```
XR_DEFINE_HANDLE(XrGeometryInstanceFB)
```

[XrGeometryInstanceFB](#) represents a geometry instance used in a passthrough layer.

## New Flag Types

```
typedef XrFlags64 XrPassthroughFlagsFB;
```

Specify additional creation behavior.

```
// Flag bits for XrPassthroughFlagsFB
static const XrPassthroughFlagsFB XR_PASSTHROUGH_IS_RUNNING_AT_CREATION_BIT_FB =
0x00000001;
static const XrPassthroughFlagsFB XR_PASSTHROUGH_LAYER_DEPTH_BIT_FB = 0x00000002;
```

## Flag Descriptions

- **XR\_PASSTHROUGH\_IS\_RUNNING\_AT\_CREATION\_BIT\_FB** — The object (passthrough, layer) is running at creation.
- **XR\_PASSTHROUGH\_LAYER\_DEPTH\_BIT\_FB** — The passthrough system sends depth information to the compositor. Only applicable to layer objects.

```
typedef XrFlags64 XrPassthroughStateChangedFlagsFB;
```

Specify additional state change behavior.

```
// Flag bits for XrPassthroughStateChangedFlagsFB
static const XrPassthroughStateChangedFlagsFB
XR_PASSTHROUGH_STATE_CHANGED_REINIT_REQUIRED_BIT_FB = 0x00000001;
static const XrPassthroughStateChangedFlagsFB
XR_PASSTHROUGH_STATE_CHANGED_NON_RECOVERABLE_ERROR_BIT_FB = 0x00000002;
static const XrPassthroughStateChangedFlagsFB
XR_PASSTHROUGH_STATE_CHANGED_RECOVERABLE_ERROR_BIT_FB = 0x00000004;
static const XrPassthroughStateChangedFlagsFB
XR_PASSTHROUGH_STATE_CHANGED_RESTORED_ERROR_BIT_FB = 0x00000008;
```

## Flag Descriptions

- **XR\_PASSTHROUGH\_STATE\_CHANGED\_REINIT\_REQUIRED\_BIT\_FB** — Passthrough system requires reinitialization.
- **XR\_PASSTHROUGH\_STATE\_CHANGED\_NON\_RECOVERABLE\_ERROR\_BIT\_FB** — Non-recoverable error has occurred. A device reboot or a firmware update may be required.
- **XR\_PASSTHROUGH\_STATE\_CHANGED\_RECOVERABLE\_ERROR\_BIT\_FB** — A recoverable error has occurred. The runtime will attempt to recover, but some functionality may be temporarily unavailable.
- **XR\_PASSTHROUGH\_STATE\_CHANGED\_RESTORED\_ERROR\_BIT\_FB** — The runtime has recovered from a previous error and is functioning normally.

```
typedef XrFlags64 XrPassthroughCapabilityFlagsFB;
```

Specify passthrough system capabilities.

```
// Flag bits for XrPassthroughCapabilityFlagsFB
static const XrPassthroughCapabilityFlagsFB XR_PASSTHROUGH_CAPABILITY_BIT_FB =
0x00000001;
static const XrPassthroughCapabilityFlagsFB XR_PASSTHROUGH_CAPABILITY_COLOR_BIT_FB =
0x00000002;
static const XrPassthroughCapabilityFlagsFB XR_PASSTHROUGH_CAPABILITY_LAYER_DEPTH_BIT_FB
= 0x00000004;
```

## Flag Descriptions

- `XR_PASSTHROUGH_CAPABILITY_BIT_FB` — The system supports passthrough.
- `XR_PASSTHROUGH_CAPABILITY_COLOR_BIT_FB` — The system can show passthrough with realistic colors. `XR_PASSTHROUGH_CAPABILITY_BIT_FB` **must** be set if `XR_PASSTHROUGH_CAPABILITY_COLOR_BIT_FB` is set.
- `XR_PASSTHROUGH_CAPABILITY_LAYER_DEPTH_BIT_FB` — The system supports passthrough layers composited using depth testing. `XR_PASSTHROUGH_CAPABILITY_BIT_FB` **must** be set if `XR_PASSTHROUGH_CAPABILITY_LAYER_DEPTH_BIT_FB` is set.

## New Enum Constants

- `XR_PASSTHROUGH_COLOR_MAP_MONO_SIZE_FB`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SYSTEM_PASSTHROUGH_PROPERTIES_FB`
- `XR_TYPE_PASSTHROUGH_CREATE_INFO_FB`
- `XR_TYPE_PASSTHROUGH_LAYER_CREATE_INFO_FB`
- `XR_TYPE_COMPOSITION_LAYER_PASSTHROUGH_FB`
- `XR_TYPE_GEOMETRY_INSTANCE_CREATE_INFO_FB`
- `XR_TYPE_GEOMETRY_INSTANCE_TRANSFORM_FB`
- `XR_TYPE_PASSTHROUGH_STYLE_FB`
- `XR_TYPE_PASSTHROUGH_COLOR_MAP_MONO_TO_RGBA_FB`
- `XR_TYPE_PASSTHROUGH_COLOR_MAP_MONO_TO_MONO_FB`
- `XR_TYPE_PASSTHROUGH_BRIGHTNESS_CONTRAST_SATURATION_FB`
- `XR_TYPE_EVENT_DATA_PASSTHROUGH_STATE_CHANGED_FB`



[XrResult](#) enumeration is extended with:

- **XR\_ERROR\_UNEXPECTED\_STATE\_PASSTHROUGH\_FB** The state of an object for which a function is called is not one of the expected states for that function.
- **XR\_ERROR\_FEATURE\_ALREADY\_CREATED\_PASSTHROUGH\_FB** An application attempted to create a feature when one has already been created and only one can exist.
- **XR\_ERROR\_FEATURE\_REQUIRED\_PASSTHROUGH\_FB** A feature is required before the function can be called.
- **XR\_ERROR\_NOT\_PERMITTED\_PASSTHROUGH\_FB** Operation is not permitted.
- **XR\_ERROR\_INSUFFICIENT\_RESOURCES\_PASSTHROUGH\_FB** The runtime does not have sufficient resources to perform the operation. Either the object being created is too large, or too many objects of a specific kind have been created.

## New Enums

Specify the kind of passthrough behavior the layer provides.

```
typedef enum XrPassthroughLayerPurposeFB {
    XR_PASSTHROUGH_LAYER_PURPOSE_RECONSTRUCTION_FB = 0,
    XR_PASSTHROUGH_LAYER_PURPOSE_PROJECTED_FB = 1,
    // Provided by XR_FB_passthrough_keyboard_hands
    XR_PASSTHROUGH_LAYER_PURPOSE_TRACKED_KEYBOARD_HANDS_FB = 1000203001,
    // Provided by XR_FB_passthrough_keyboard_hands
    XR_PASSTHROUGH_LAYER_PURPOSE_TRACKED_KEYBOARD_MASKED_HANDS_FB = 1000203002,
    XR_PASSTHROUGH_LAYER_PURPOSE_MAX_ENUM_FB = 0x7FFFFFFF
} XrPassthroughLayerPurposeFB;
```

## Enumerant Descriptions

- **XR\_PASSTHROUGH\_LAYER\_PURPOSE\_RECONSTRUCTION\_FB** — Reconstruction passthrough (full screen environment)
- **XR\_PASSTHROUGH\_LAYER\_PURPOSE\_PROJECTED\_FB** — Projected passthrough (using a custom surface)
- **XR\_PASSTHROUGH\_LAYER\_PURPOSE\_TRACKED\_KEYBOARD\_HANDS\_FB** — Passthrough layer purpose for keyboard hands presence. (Added by the [XR\\_FB\\_passthrough\\_keyboard\\_hands](#) extension)
- **XR\_PASSTHROUGH\_LAYER\_PURPOSE\_TRACKED\_KEYBOARD\_MASKED\_HANDS\_FB** — Passthrough layer purpose for keyboard hands presence with keyboard masked hand transitions (i.e passthrough hands rendered only when they are over the keyboard). (Added by the [XR\\_FB\\_passthrough\\_keyboard\\_hands](#) extension)

## New Structures

The [XrSystemPassthroughPropertiesFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrSystemPassthroughPropertiesFB {
    XrStructureType    type;
    const void*        next;
    XrBool32           supportsPassthrough;
} XrSystemPassthroughPropertiesFB;
```

It describes a passthrough system property.

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsPassthrough` defines whether the system supports the passthrough feature.

### Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to using [XrSystemPassthroughPropertiesFB](#)
- `type` **must** be `XR_TYPE_SYSTEM_PASSTHROUGH_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Structures

The [XrSystemPassthroughProperties2FB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrSystemPassthroughProperties2FB {
    XrStructureType    type;
    const void*        next;
    XrPassthroughCapabilityFlagsFB capabilities;
} XrSystemPassthroughProperties2FB;
```

Applications **can** pass this structure in a call to [xrGetSystemProperties](#) to query passthrough system properties. Applications **should** verify that the runtime implements [XR\\_FB\\_passthrough](#) spec version 3 or newer before doing so. In older versions, this structure is not supported and will be left unpopulated. Applications **should** use [XrSystemPassthroughPropertiesFB](#) in that case.

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **capabilities** defines a set of features supported by the passthrough system.

### Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to using [XrSystemPassthroughProperties2FB](#)
- **type** **must** be `XR_TYPE_SYSTEM_PASSTHROUGH_PROPERTIES2_FB`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrPassthroughCreateInfoFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrPassthroughCreateInfoFB {
    XrStructureType    type;
    const void*        next;
    XrPassthroughFlagsFB flags;
} XrPassthroughCreateInfoFB;
```

It contains parameters used to specify a new passthrough feature.

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **flags** is a bitmask of [XrPassthroughFlagBitsFB](#) that specify additional behavior.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using `XrPassthroughCreateInfoFB`
- `type` **must** be `XR_TYPE_PASSTHROUGH_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `flags` **must** be a valid combination of `XrPassthroughFlagBitsFB` values
- `flags` **must** not be `0`

The `XrPassthroughLayerCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrPassthroughLayerCreateInfoFB {
    XrStructureType          type;
    const void*              next;
    XrPassthroughFB         passthrough;
    XrPassthroughFlagsFB    flags;
    XrPassthroughLayerPurposeFB purpose;
} XrPassthroughLayerCreateInfoFB;
```

It contains parameters used to specify a new passthrough layer.

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `passthrough` an `XrPassthroughFB` handle.
- `flags` `XrPassthroughFlagsFB` that specify additional behavior.
- `purpose` `XrPassthroughLayerPurposeFB` that specifies the layer's purpose.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using `XrPassthroughLayerCreateInfoFB`
- `type` **must** be `XR_TYPE_PASSTHROUGH_LAYER_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `passthrough` **must** be a valid `XrPassthroughFB` handle
- `flags` **must** be a valid combination of `XrPassthroughFlagBitsFB` values
- `flags` **must** not be `0`
- `purpose` **must** be a valid `XrPassthroughLayerPurposeFB` value

The `XrCompositionLayerPassthroughFB` structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrCompositionLayerPassthroughFB {
    XrStructureType      type;
    const void*          next;
    XrCompositionLayerFlags flags;
    XrSpace              space;
    XrPassthroughLayerFB layerHandle;
} XrCompositionLayerPassthroughFB;
```

It is a composition layer type that may be submitted in `xrEndFrame` where an `XrCompositionLayerBaseHeader` is specified, as a stand-in for the actual passthrough contents.

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of `XrCompositionLayerFlagBits` that specify additional behavior.
- `space` is the `XrSpace` that specifies the layer's space - **must** be `XR_NULL_HANDLE`.
- `layerHandle` is the `XrPassthroughLayerFB` that defines this layer's behavior.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using `XrCompositionLayerPassthroughFB`
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_PASSTHROUGH_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `flags` **must** be a valid combination of `XrCompositionLayerFlagBits` values
- `flags` **must** not be `0`
- `space` **must** be a valid `XrSpace` handle
- `layerHandle` **must** be a valid `XrPassthroughLayerFB` handle
- Both of `layerHandle` and `space` **must** have been created, allocated, or retrieved from the same `XrSession`

The `XrGeometryInstanceCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrGeometryInstanceCreateInfoFB {
    XrStructureType      type;
    const void*          next;
    XrPassthroughLayerFB layer;
    XrTriangleMeshFB    mesh;
    XrSpace               baseSpace;
    XrPosef               pose;
    XrVector3f            scale;
} XrGeometryInstanceCreateInfoFB;
```

It contains parameters to specify a new geometry instance.

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layer` is the [XrPassthroughLayerFB](#).
- `mesh` is the [XrTriangleMeshFB](#).
- `baseSpace` is the [XrSpace](#) that defines the geometry instance's base space for transformations.
- `pose` is the [XrPosef](#) that defines the geometry instance's pose.
- `scale` is the [XrVector3f](#) that defines the geometry instance's scale.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to using [XrGeometryInstanceCreateInfoFB](#)
- `type` **must** be `XR_TYPE_GEOMETRY_INSTANCE_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layer` **must** be a valid [XrPassthroughLayerFB](#) handle
- `mesh` **must** be a valid [XrTriangleMeshFB](#) handle
- `baseSpace` **must** be a valid [XrSpace](#) handle
- Each of `baseSpace`, `layer`, and `mesh` **must** have been created, allocated, or retrieved from the same [XrSession](#)

The [XrGeometryInstanceTransformFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrGeometryInstanceTransformFB {
    XrStructureType    type;
    const void*        next;
    XrSpace             baseSpace;
    XrTime              time;
    XrPosef             pose;
    XrVector3f          scale;
} XrGeometryInstanceTransformFB;
```

It describes a transformation for a geometry instance.

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **baseSpace** is the [XrSpace](#) that defines the geometry instance's base space for transformations.
- **time** is the [XrTime](#) that define the time at which the transform is applied.
- **pose** is the [XrPosef](#) that defines the geometry instance's pose.
- **scale** is the [XrVector3f](#) that defines the geometry instance's scale.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to using [XrGeometryInstanceTransformFB](#)
- **type** **must** be `XR_TYPE_GEOMETRY_INSTANCE_TRANSFORM_FB`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **baseSpace** **must** be a valid [XrSpace](#) handle

The [XrPassthroughStyleFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrPassthroughStyleFB {
    XrStructureType    type;
    const void*        next;
    float               textureOpacityFactor;
    XrColor4f           edgeColor;
} XrPassthroughStyleFB;
```



## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `textureOpacityFactor` is the opacity of the passthrough imagery in the range [0, 1].
- `edgeColor` is the `XrColor4f` that defines the edge rendering color. Edges are detected in the original passthrough imagery and rendered on top of it. Edge rendering is disabled when the alpha value of `edgeColor` is zero.

`XrPassthroughStyleFB` lets applications customize the appearance of passthrough layers. In addition to the parameters specified here, applications **may** add one of the following structures to the structure chain: `XrPassthroughColorMapMonoToRgbaFB`, `XrPassthroughColorMapMonoToMonoFB`, `XrPassthroughBrightnessContrastSaturationFB`. These structures are mutually exclusive. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if more than one of them are present in the structure chain.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using `XrPassthroughStyleFB`
- `type` **must** be `XR_TYPE_PASSTHROUGH_STYLE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: `XrPassthroughBrightnessContrastSaturationFB`, `XrPassthroughColorMapInterpolatedLutMETA`, `XrPassthroughColorMapLutMETA`, `XrPassthroughColorMapMonoToMonoFB`, `XrPassthroughColorMapMonoToRgbaFB`

The `XrPassthroughColorMapMonoToRgbaFB` structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrPassthroughColorMapMonoToRgbaFB {
    XrStructureType    type;
    const void*        next;
    XrColor4f          textureColorMap[XR_PASSTHROUGH_COLOR_MAP_MONO_SIZE_FB];
} XrPassthroughColorMapMonoToRgbaFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `textureColorMap` is an array of [XrColor4f](#) colors to which the passthrough imagery luminance values are mapped.

[XrPassthroughColorMapMonoToRgbaFB](#) lets applications define a map which replaces each input luminance value in the passthrough imagery with an RGBA color value. The map is applied before any additional effects (such as edges) are rendered on top.

[XrPassthroughColorMapMonoToRgbaFB](#) is provided in the `next` chain of [XrPassthroughStyleFB](#).

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using [XrPassthroughColorMapMonoToRgbaFB](#)
- `type` **must** be `XR_TYPE_PASSTHROUGH_COLOR_MAP_MONO_TO_RGBA_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrPassthroughColorMapMonoToMonoFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrPassthroughColorMapMonoToMonoFB {
    XrStructureType    type;
    const void*        next;
    uint8_t            textureColorMap[XR_PASSTHROUGH_COLOR_MAP_MONO_SIZE_FB];
} XrPassthroughColorMapMonoToMonoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `textureColorMap` is an array of `uint8_t` grayscale color values to which the passthrough luminance values are mapped.

[XrPassthroughColorMapMonoToMonoFB](#) lets applications define a map which replaces each input

luminance value in the passthrough imagery with a grayscale color value defined in `textureColorMap`. The map is applied before any additional effects (such as edges) are rendered on top.

`XrPassthroughColorMapMonoToMonoFB` is provided in the `next` chain of `XrPassthroughStyleFB`.

### Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using `XrPassthroughColorMapMonoToMonoFB`
- `type` **must** be `XR_TYPE_PASSTHROUGH_COLOR_MAP_MONO_TO_MONO_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

The `XrPassthroughBrightnessContrastSaturationFB` structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrPassthroughBrightnessContrastSaturationFB {
    XrStructureType    type;
    const void*        next;
    float               brightness;
    float               contrast;
    float               saturation;
} XrPassthroughBrightnessContrastSaturationFB;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `brightness` is the brightness adjustment value in the range [-100, 100]. The neutral element is 0.
- `contrast` is the contrast adjustment value in the range [0, Infinity]. The neutral element is 1.
- `saturation` is the saturation adjustment value in the range [0, Infinity]. The neutral element is 1.

`XrPassthroughBrightnessContrastSaturationFB` lets applications adjust the brightness, contrast, and saturation of passthrough layers. The adjustments only are applied before any additional effects (such as edges) are rendered on top.

The adjustments are applied in CIELAB color space (white point D65) using the following formulas:

- $L^* = \text{clamp}((L^* - 50) \times \text{contrast} + 50, 0, 100)$
- $L^{*'} = \text{clamp}(L^* + \text{brightness}, 0, 100)$
- $(a^{*'}, b^{*'}) = (a^*, b^*) \times \text{saturation}$
- Resulting color:  $(L^{*'}, a^{*'}, b^{*'})$

[XrPassthroughBrightnessContrastSaturationFB](#) is provided in the `next` chain of [XrPassthroughStyleFB](#).

### Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using [XrPassthroughBrightnessContrastSaturationFB](#)
- `type` **must** be `XR_TYPE_PASSTHROUGH_BRIGHTNESS_CONTRAST_SATURATION_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataPassthroughStateChangedFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough
typedef struct XrEventDataPassthroughStateChangedFB {
    XrStructureType          type;
    const void*              next;
    XrPassthroughStateChangedFlagsFB flags;
} XrEventDataPassthroughStateChangedFB;
```

It describes an event data for state changes return by [xrPollEvent](#).

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` [XrPassthroughStateChangedFlagsFB](#) that specify additional behavior.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to using `XrEventDataPassthroughStateChangedFB`
- `type` **must** be `XR_TYPE_EVENT_DATA_PASSTHROUGH_STATE_CHANGED_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `flags` **must** be a valid combination of `XrPassthroughStateChangedFlagBitsFB` values
- `flags` **must** not be `0`

## New Functions

The `xrCreatePassthroughFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrCreatePassthroughFB(
    XrSession session,
    const XrPassthroughCreateInfoFB* createInfo,
    XrPassthroughFB* outPassthrough);
```

## Parameter Descriptions

- `session` is the `XrSession`.
- `createInfo` is the `XrPassthroughCreateInfoFB`.
- `outPassthrough` is the `XrPassthroughFB`.

Creates an `XrPassthroughFB` handle. The returned passthrough handle **may** be subsequently used in API calls.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to calling `xrCreatePassthroughFB`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrPassthroughCreateInfoFB` structure
- `outPassthrough` **must** be a pointer to an `XrPassthroughFB` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_UNKNOWN_PASSTHROUGH_FB`
- `XR_ERROR_NOT_PERMITTED_PASSTHROUGH_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_FEATURE_ALREADY_CREATED_PASSTHROUGH_FB`

The `xrDestroyPassthroughFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrDestroyPassthroughFB(
    XrPassthroughFB                passthrough);
```

## Parameter Descriptions

- `passthrough` is the `XrPassthroughFB` to be destroyed.

Destroys an `XrPassthroughFB` handle.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to calling `xrDestroyPassthroughFB`
- `passthrough` **must** be a valid `XrPassthroughFB` handle

## Thread Safety

- Access to `passthrough`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrPassthroughStartFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrPassthroughStartFB(
    XrPassthroughFB          passthrough);
```

## Parameter Descriptions

- `passthrough` is the `XrPassthroughFB` to be started.

Starts an `XrPassthroughFB` feature. If the feature is not started, either explicitly with a call to `xrPassthroughStartFB`, or implicitly at creation using the behavior flags, it is considered paused. When the feature is paused, runtime will stop rendering and compositing all passthrough layers produced on behalf of the application, and may free up some or all the resources used to produce passthrough until `xrPassthroughStartFB` is called.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to calling `xrPassthroughStartFB`
- `passthrough` **must** be a valid `XrPassthroughFB` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrPassthroughPauseFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrPassthroughPauseFB(
    XrPassthroughFB          passthrough);
```

## Parameter Descriptions

- `passthrough` is the `XrPassthroughFB` to be paused.

Pauses an `XrPassthroughFB` feature. When the feature is paused, runtime will stop rendering and compositing all passthrough layers produced on behalf of the application, and may free up some or all the resources used to produce passthrough until `xrPassthroughStartFB` is called.



## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to calling `xrPassthroughPauseFB`
- `passthrough` **must** be a valid `XrPassthroughFB` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrCreatePassthroughLayerFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrCreatePassthroughLayerFB(
    XrSession session,
    const XrPassthroughLayerCreateInfoFB* createInfo,
    XrPassthroughLayerFB* outLayer);
```

## Parameter Descriptions

- `session` is the `XrSession`.
- `createInfo` is the `XrPassthroughLayerCreateInfoFB`.
- `outLayer` is the `XrPassthroughLayerFB`.

Creates an [XrPassthroughLayerFB](#) handle. The returned layer handle **may** be subsequently used in API calls. Layer objects may be used to specify rendering properties of the layer, such as styles, and compositing rules.

### Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to calling [xrCreatePassthroughLayerFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrPassthroughLayerCreateInfoFB](#) structure
- `outLayer` **must** be a pointer to an [XrPassthroughLayerFB](#) handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_UNKNOWN_PASSTHROUGH_FB`
- `XR_ERROR_INSUFFICIENT_RESOURCES_PASSTHROUGH_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_FEATURE_REQUIRED_PASSTHROUGH_FB`

The [xrDestroyPassthroughLayerFB](#) function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrDestroyPassthroughLayerFB(
    XrPassthroughLayerFB          layer);
```

## Parameter Descriptions

- **layer** is the [XrPassthroughLayerFB](#) to be destroyed.

Destroys an [XrPassthroughLayerFB](#) handle.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to calling [xrDestroyPassthroughLayerFB](#)
- **layer** **must** be a valid [XrPassthroughLayerFB](#) handle

## Thread Safety

- Access to **layer**, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- [XR\\_SUCCESS](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

The [xrPassthroughLayerPauseFB](#) function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrPassthroughLayerPauseFB(
    XrPassthroughLayerFB          layer);
```

## Parameter Descriptions

- **layer** is the [XrPassthroughLayerFB](#) to be paused.

Pauses an [XrPassthroughLayerFB](#) layer. Runtime will not render or composite paused layers.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to calling [xrPassthroughLayerPauseFB](#)
- **layer** **must** be a valid [XrPassthroughLayerFB](#) handle

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_UNEXPECTED\\_STATE\\_PASSTHROUGH\\_FB](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

The [xrPassthroughLayerResumeFB](#) function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrPassthroughLayerResumeFB(
    XrPassthroughLayerFB          layer);
```

## Parameter Descriptions

- **layer** is the [XrPassthroughLayerFB](#) to be resumed.

Resumes an [XrPassthroughLayerFB](#) layer.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to calling [xrPassthroughLayerResumeFB](#)
- **layer** **must** be a valid [XrPassthroughLayerFB](#) handle

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_UNEXPECTED\\_STATE\\_PASSTHROUGH\\_FB](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

The [xrPassthroughLayerSetStyleFB](#) function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrPassthroughLayerSetStyleFB(
    XrPassthroughLayerFB          layer,
    const XrPassthroughStyleFB*   style);
```

## Parameter Descriptions

- **layer** is the [XrPassthroughLayerFB](#) to get the style.
- **style** is the [XrPassthroughStyleFB](#) to be set.

Sets an [XrPassthroughStyleFB](#) style on an [XrPassthroughLayerFB](#) layer.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to calling [xrPassthroughLayerSetStyleFB](#)
- **layer** **must** be a valid [XrPassthroughLayerFB](#) handle
- **style** **must** be a pointer to a valid [XrPassthroughStyleFB](#) structure

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

The [xrCreateGeometryInstanceFB](#) function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrCreateGeometryInstanceFB(
    XrSession session,
    const XrGeometryInstanceCreateInfoFB* createInfo,
    XrGeometryInstanceFB* outGeometryInstance);
```

## Parameter Descriptions

- `session` is the [XrSession](#).
- `createInfo` is the [XrGeometryInstanceCreateInfoFB](#).
- `outGeometryInstance` is the [XrGeometryInstanceFB](#).

Creates an [XrGeometryInstanceFB](#) handle. Geometry instance functionality requires [XR\\_FB\\_triangle\\_mesh](#) extension to be enabled. An [XrGeometryInstanceFB](#) connects a layer, a mesh, and a transformation, with the semantics that a specific mesh will be instantiated in a specific layer with a specific transformation. A mesh can be instantiated multiple times, in the same or in different layers.

## Valid Usage (Implicit)

- The [XR\\_FB\\_passthrough](#) extension **must** be enabled prior to calling [xrCreateGeometryInstanceFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrGeometryInstanceCreateInfoFB](#) structure
- `outGeometryInstance` **must** be a pointer to an [XrGeometryInstanceFB](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_INSUFFICIENT_RESOURCES_PASSTHROUGH_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrDestroyGeometryInstanceFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrDestroyGeometryInstanceFB(
    XrGeometryInstanceFB          instance);
```

## Parameter Descriptions

- `instance` is the `XrGeometryInstanceFB` to be destroyed.

Destroys an `XrGeometryInstanceFB` handle. Destroying an `XrGeometryInstanceFB` does not destroy a mesh and does not free mesh resources. Destroying a layer invalidates all geometry instances attached to it. Destroying a mesh invalidates all its instances.



## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to calling `xrDestroyGeometryInstanceFB`
- `instance` **must** be a valid `XrGeometryInstanceFB` handle

## Thread Safety

- Access to `instance`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGeometryInstanceSetTransformFB` function is defined as:

```
// Provided by XR_FB_passthrough
XrResult xrGeometryInstanceSetTransformFB(
    XrGeometryInstanceFB          instance,
    const XrGeometryInstanceTransformFB* transformation);
```

## Parameter Descriptions

- `instance` is the `XrGeometryInstanceFB` to get the transform.
- `transformation` is the `XrGeometryInstanceTransformFB` to be set.

Sets an `XrGeometryInstanceTransformFB` transform on an `XrGeometryInstanceFB` geometry instance.

## Valid Usage (Implicit)

- The `XR_FB_passthrough` extension **must** be enabled prior to calling `xrGeometryInstanceSetTransformFB`
- `instance` **must** be a valid `XrGeometryInstanceFB` handle
- `transformation` **must** be a pointer to a valid `XrGeometryInstanceTransformFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_TIME_INVALID`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

### Version History

- Revision 1, 2021-09-01 (Anton Vaneev)
  - Initial extension description
- Revision 2, 2022-03-16 (Johannes Schmid)
  - Introduce `XrPassthroughBrightnessContrastSaturationFB`.
  - Revise the documentation of `XrPassthroughStyleFB` and its descendants.
- Revision 3, 2022-07-14 (Johannes Schmid)
  - Introduce a new struct for querying passthrough system capabilities: `XrSystemPassthroughProperties2FB`.

- Introduce a new flag bit that enables submission of depth maps for compositing:  
`XR_PASSTHROUGH_LAYER_DEPTH_BIT_FB`.

## 12.64. `XR_FB_passthrough_keyboard_hands`

### Name String

`XR_FB_passthrough_keyboard_hands`

### Extension Type

Instance extension

### Registered Extension Number

204

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_FB\\_passthrough](#)

### Contributors

Ante Trbojevic, Facebook  
Cass Everitt, Facebook  
Federico Schliemann, Facebook  
Anton Vaneev, Facebook  
Johannes Schmid, Facebook

### Overview

This extension enables applications to show passthrough hands when hands are placed over the tracked keyboard. It enables users to see their hands over the keyboard in a mixed reality application. This extension is dependent on [XR\\_FB\\_passthrough](#) extension which can be used to create a passthrough layer for hand presence use-case.

The extension supports a single pair of hands (one left and one right hand), multiple pair of hands are not supported.

This extension allows:

- Creation of keyboard hands passthrough layer using [xrCreatePassthroughLayerFB](#)
- Setting the level of intensity for the hand mask in a passthrough layer with purpose [XrPassthroughLayerPurposeFB](#) as `XR_PASSTHROUGH_LAYER_PURPOSE_TRACKED_KEYBOARD_HANDBS_FB` or `XR_PASSTHROUGH_LAYER_PURPOSE_TRACKED_KEYBOARD_MASKED_HANDBS_FB`

## New Enum Constants

[XrPassthroughLayerPurposeFB](#) enumeration is extended with a new constant:

- [XR\\_PASSTHROUGH\\_LAYER\\_PURPOSE\\_TRACKED\\_KEYBOARD\\_HANDS\\_FB](#) - It defines a keyboard hands presence purpose of passthrough layer (i.e. basic mode, without hand transitions).
- [XR\\_PASSTHROUGH\\_LAYER\\_PURPOSE\\_TRACKED\\_KEYBOARD\\_MASKED\\_HANDS\\_FB](#) - It defines a keyboard hands presence purpose of passthrough layer with keyboard masked hand transitions. A hand mask will be visible only when hands are inside the region of VR keyboard (i.e. hands over the keyboard).

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_PASSTHROUGH\\_KEYBOARD\\_HANDS\\_INTENSITY\\_FB](#)

## New Structures

The [XrPassthroughKeyboardHandsIntensityFB](#) structure is defined as:

```
// Provided by XR_FB_passthrough_keyboard_hands
typedef struct XrPassthroughKeyboardHandsIntensityFB {
    XrStructureType    type;
    const void*        next;
    float               leftHandIntensity;
    float               rightHandIntensity;
} XrPassthroughKeyboardHandsIntensityFB;
```

### Member Descriptions

- [type](#) is the [XrStructureType](#) of this structure.
- [next](#) is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- [leftHandIntensity](#) defines an intensity for the left tracked hand.
- [rightHandIntensity](#) defines an intensity for the right tracked hand.

[XrPassthroughKeyboardHandsIntensityFB](#) describes intensities of passthrough hands, and is used as a parameter to [xrPassthroughLayerSetKeyboardHandsIntensityFB](#).

Each of the intensity values [leftHandIntensity](#) and [rightHandIntensity](#) **must** be in the range [0.0, 1.0]. The hand intensity value represents the level of visibility of rendered hand, the minimal value of the intensity 0.0 represents the fully transparent hand (not visible), the maximal value of 1.0 represented fully opaque hands (maximal visibility).

If either `leftHandIntensity` or `rightHandIntensity` is outside the range [0.0, 1.0], the runtime must return `XR_ERROR_VALIDATION_FAILURE`.

### Valid Usage (Implicit)

- The `XR_FB_passthrough_keyboard_hands` extension **must** be enabled prior to using `XrPassthroughKeyboardHandsIntensityFB`
- `type` **must** be `XR_TYPE_PASSTHROUGH_KEYBOARD_HANDS_INTENSITY_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### New Functions

The `xrPassthroughLayerSetKeyboardHandsIntensityFB` function is defined as:

```
// Provided by XR_FB_passthrough_keyboard_hands
XrResult xrPassthroughLayerSetKeyboardHandsIntensityFB(
    XrPassthroughLayerFB layer,
    const XrPassthroughKeyboardHandsIntensityFB* intensity);
```

### Parameter Descriptions

- `layer` is the `XrPassthroughLayerFB` to apply the intensity.
- `intensity` is the `XrPassthroughKeyboardHandsIntensityFB` to be set.

Sets an `XrPassthroughKeyboardHandsIntensityFB` intensity on an `XrPassthroughLayerFB` layer.

### Valid Usage (Implicit)

- The `XR_FB_passthrough_keyboard_hands` extension **must** be enabled prior to calling `xrPassthroughLayerSetKeyboardHandsIntensityFB`
- `layer` **must** be a valid `XrPassthroughLayerFB` handle
- `intensity` **must** be a pointer to a valid `XrPassthroughKeyboardHandsIntensityFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

### Version History

- Revision 1, 2021-11-23 (Ante Trbojevic)
  - Initial extension description
- Revision 2, 2022-03-16 (Ante Trbojevic)
  - Introduce `XR_PASSTHROUGH_LAYER_PURPOSE_TRACKED_KEYBOARD_MASKED_HANDS_FB`

## 12.65. XR\_FB\_render\_model

### Name String

`XR_FB_render_model`

### Extension Type

Instance extension

### Registered Extension Number

120

### Revision

4

### Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Leonard Tsai, Meta  
Xiang Wei, Meta  
Robert Memmott, Meta

## Overview

This extension allows applications to request GLTF models for certain connected devices supported by the runtime. Paths that correspond to these devices will be provided through the extension and can be used to get information about the models as well as loading them.

## New Flag Types

```
typedef XrFlags64 XrRenderModelFlagsFB;
```

```
// Flag bits for XrRenderModelFlagsFB
static const XrRenderModelFlagsFB XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_1_BIT_FB =
0x00000001;
static const XrRenderModelFlagsFB XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_2_BIT_FB =
0x00000002;
```

### Flag Descriptions

- `XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_1_BIT_FB` — Minimal level of support. Can only contain a single mesh. Can only contain a single texture. Can not contain transparency. Assumes unlit rendering. Requires Extension `KHR_texturebasisu`.
- `XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_2_BIT_FB` — All of `XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_1_BIT_FB` support plus: Multiple meshes. Multiple Textures. Texture Transparency.

Render Model Support Levels: An application **should** request a model of a certain complexity via the `XrRenderModelCapabilitiesRequestFB` on the structure chain of `XrRenderModelPropertiesFB` passed into `xrGetRenderModelPropertiesFB`. The flags on the `XrRenderModelCapabilitiesRequestFB` are an acknowledgement of the application's ability to render such a model. Multiple values of `XrRenderModelFlagBitsFB` can be set on this variable to indicate acceptance of different support levels. The flags parameter on the `XrRenderModelPropertiesFB` will indicate what capabilities the model in the runtime actually requires. It will be set to a single value of `XrRenderModelFlagBitsFB`.

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SYSTEM\\_RENDER\\_MODEL\\_PROPERTIES\\_FB](#)
- [XR\\_TYPE\\_RENDER\\_MODEL\\_PATH\\_INFO\\_FB](#)
- [XR\\_TYPE\\_RENDER\\_MODEL\\_PROPERTIES\\_FB](#)
- [XR\\_TYPE\\_RENDER\\_MODEL\\_BUFFER\\_FB](#)
- [XR\\_TYPE\\_RENDER\\_MODEL\\_LOAD\\_INFO\\_FB](#)
- [XR\\_MAX\\_RENDER\\_MODEL\\_NAME\\_SIZE\\_FB](#)

## New Defines

```
// Provided by XR_FB_render_model
#define XR_NULL_RENDER_MODEL_KEY_FB 0
```

[XR\\_NULL\\_RENDER\\_MODEL\\_KEY\\_FB](#) defines an invalid model key atom.

## New Base Types

```
// Provided by XR_FB_render_model
XR_DEFINE_ATOM(XrRenderModelKeyFB)
```

The unique model key used to retrieve the data for the render model that is valid across multiple instances and installs. The application can use this key along with the model version to update its cached or saved version of the model.

## New Structures

The [XrSystemRenderModelPropertiesFB](#) structure is defined as:

```
// Provided by XR_FB_render_model
typedef struct XrSystemRenderModelPropertiesFB {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsRenderModelLoading;
} XrSystemRenderModelPropertiesFB;
```



It describes a render model system property.

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsRenderModelLoading` defines whether the system supports loading render models.

### Valid Usage (Implicit)

- The `XR_FB_render_model` extension **must** be enabled prior to using [XrSystemRenderModelPropertiesFB](#)
- `type` **must** be `XR_TYPE_SYSTEM_RENDER_MODEL_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrRenderModelPathInfoFB](#) structure is defined as:

```
// Provided by XR_FB_render_model
typedef struct XrRenderModelPathInfoFB {
    XrStructureType    type;
    void*              next;
    XrPath              path;
} XrRenderModelPathInfoFB;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `path` is a valid [XrPath](#) used for retrieving model properties from [xrGetRenderModelPropertiesFB](#).

[XrRenderModelPathInfoFB](#) contains a model path supported by the device when returned from [xrEnumerateRenderModelPathsFB](#). This path can be used to request information about the render model for the connected device that the path represents using [xrGetRenderModelPropertiesFB](#).

## Possible Render Model Paths

- Controller models with origin at the grip pose.
  - `/model_fb/controller/left`
  - `/model_fb/controller/right`
- Keyboard models with origin at the center of its bounding box.
  - `/model_fb/keyboard/local`
  - `/model_fb/keyboard/remote`
  - `/model_meta/keyboard/virtual`  
(if the `XR_META_virtual_keyboard` extension is enabled)

## Valid Usage (Implicit)

- The `XR_FB_render_model` extension **must** be enabled prior to using `XrRenderModelPathInfoFB`
- `type` **must** be `XR_TYPE_RENDER_MODEL_PATH_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrRenderModelPropertiesFB` structure is defined as:

```
// Provided by XR_FB_render_model
typedef struct XrRenderModelPropertiesFB {
    XrStructureType      type;
    void*                next;
    uint32_t             vendorId;
    char                 modelName[XR_MAX_RENDER_MODEL_NAME_SIZE_FB];
    XrRenderModelKeyFB   modelKey;
    uint32_t             modelVersion;
    XrRenderModelFlagsFB flags;
} XrRenderModelPropertiesFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. [XrRenderModelCapabilitiesRequestFB](#) is a structure in this structure chain and **should** be linked when this structure is passed to [xrGetRenderModelPropertiesFB](#).
- `vendorId` is the vendor id of the model.
- `modelName` is the name of the model.
- `modelKey` is the unique model key used to load the model in [xrLoadRenderModelFB](#).
- `modelVersion` is the version number of the model.
- `flags` is a bitmask of [XrRenderModelFlagsFB](#). After a successful call to [xrGetRenderModelPropertiesFB](#), flags must contain the support level of the model and no other support levels.

[XrRenderModelPropertiesFB](#) contains information about the render model for a device. [XrRenderModelPropertiesFB](#) **must** be provided when calling [xrGetRenderModelPropertiesFB](#). The [XrRenderModelKeyFB](#) included in the properties is a unique key for each render model that is valid across multiple instances and installs.

If the application decides to cache or save the render model in any way, `modelVersion` can be used to determine if the render model has changed. The application **should** then update its cached or saved version.

## Valid Usage (Implicit)

- The `XR_FB_render_model` extension **must** be enabled prior to using [XrRenderModelPropertiesFB](#)
- `type` **must** be `XR_TYPE_RENDER_MODEL_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrRenderModelCapabilitiesRequestFB](#)
- `modelName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_RENDER_MODEL_NAME_SIZE_FB`
- `flags` **must** be a valid combination of [XrRenderModelFlagBitsFB](#) values
- `flags` **must** not be `0`

The [XrRenderModelCapabilitiesRequestFB](#) structure is defined as:

```
// Provided by XR_FB_render_model
typedef struct XrRenderModelCapabilitiesRequestFB {
    XrStructureType      type;
    void*                next;
    XrRenderModelFlagsFB flags;
} XrRenderModelCapabilitiesRequestFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bit mask of the model complexities that the application is able to support.

[XrRenderModelCapabilitiesRequestFB](#) contains information about the render capabilities requested for a model. [XrRenderModelCapabilitiesRequestFB](#) **must** be set in the structure chain of the `next` pointer on the [XrRenderModelPropertiesFB](#) passed into the [xrGetRenderModelPropertiesFB](#) call. The `flags` on [XrRenderModelCapabilitiesRequestFB](#) represent an acknowledgement of being able to handle the individual model capability levels. If no [XrRenderModelCapabilitiesRequestFB](#) is on the structure chain then the runtime **should** treat it as if a value of `XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_1_BIT_FB` was set. If the runtime does not have a model available that matches any of the supports flags set, then it **must** return a `XR_RENDER_MODEL_UNAVAILABLE_FB` result.

## Valid Usage (Implicit)

- The [XR\\_FB\\_render\\_model](#) extension **must** be enabled prior to using [XrRenderModelCapabilitiesRequestFB](#)
- `type` **must** be `XR_TYPE_RENDER_MODEL_CAPABILITIES_REQUEST_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be a valid combination of [XrRenderModelFlagBitsFB](#) values
- `flags` **must** not be `0`

The [XrRenderModelLoadInfoFB](#) structure is defined as:

```
// Provided by XR_FB_render_model
typedef struct XrRenderModelLoadInfoFB {
    XrStructureType    type;
    void*              next;
    XrRenderModelKeyFB modelKey;
} XrRenderModelLoadInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `modelKey` is the unique model key for a connected device.

[XrRenderModelLoadInfoFB](#) is used to provide information about which render model to load. [XrRenderModelLoadInfoFB](#) **must** be provided when calling [xrLoadRenderModelFB](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_render\\_model](#) extension **must** be enabled prior to using [XrRenderModelLoadInfoFB](#)
- `type` **must** be `XR_TYPE_RENDER_MODEL_LOAD_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrRenderModelBufferFB](#) structure is defined as:

```
// Provided by XR_FB_render_model
typedef struct XrRenderModelBufferFB {
    XrStructureType    type;
    void*              next;
    uint32_t           bufferCapacityInput;
    uint32_t           bufferCountOutput;
    uint8_t*           buffer;
} XrRenderModelBufferFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `bufferCapacityInput` is the capacity of the `buffer`, or `0` to retrieve the required capacity.
- `bufferCountOutput` is the count of `uint8_t` `buffer` written, or the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an application-allocated array that will be filled with the render model binary data.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

[XrRenderModelBufferFB](#) is used when loading the binary data for a render model. [XrRenderModelBufferFB](#) **must** be provided when calling [xrLoadRenderModelFB](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_render\\_model](#) extension **must** be enabled prior to using [XrRenderModelBufferFB](#)
- `type` **must** be `XR_TYPE_RENDER_MODEL_BUFFER_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `bufferCapacityInput` is not `0`, `buffer` **must** be a pointer to an array of `bufferCapacityInput` `uint8_t` values

## New Functions

The [xrEnumerateRenderModelPathsFB](#) function is defined as:

```
// Provided by XR_FB_render_model
XrResult xrEnumerateRenderModelPathsFB(
    XrSession session,
    uint32_t pathCapacityInput,
    uint32_t* pathCountOutput,
    XrRenderModelPathInfoFB* paths);
```

## Parameter Descriptions

- `session` is the specified [XrSession](#).
- `pathCapacityInput` is the capacity of the `paths`, or 0 to retrieve the required capacity.
- `pathCountOutput` is a pointer to the count of `float paths` written, or a pointer to the required capacity in the case that `pathCapacityInput` is insufficient.
- `paths` is a pointer to an application-allocated array that will be filled with [XrRenderModelPathInfoFB](#) values that are supported by the runtime, but **can** be `NULL` if `pathCapacityInput` is 0
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `paths` size.

The application **must** call [xrEnumerateRenderModelPathsFB](#) to enumerate the valid render model paths that are supported by the runtime before calling [xrGetRenderModelPropertiesFB](#). The paths returned **may** be used later in [xrGetRenderModelPropertiesFB](#).

## Valid Usage (Implicit)

- The `XR_FB_render_model` extension **must** be enabled prior to calling [xrEnumerateRenderModelPathsFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `pathCountOutput` **must** be a pointer to a `uint32_t` value
- If `pathCapacityInput` is not 0, `paths` **must** be a pointer to an array of `pathCapacityInput` [XrRenderModelPathInfoFB](#) structures

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

The `xrGetRenderModelPropertiesFB` function is defined as:

```
// Provided by XR_FB_render_model
XrResult xrGetRenderModelPropertiesFB(
    XrSession          session,
    XrPath             path,
    XrRenderModelPropertiesFB* properties);
```

## Parameter Descriptions

- `session` is the specified `XrSession`.
- `path` is the path of the render model to get the properties for.
- `properties` is a pointer to the `XrRenderModelPropertiesFB` to write the render model information to.

`xrGetRenderModelPropertiesFB` is used for getting information for a render model using a path retrieved from `xrEnumerateRenderModelPathsFB`. The information returned will be for the connected device that corresponds to the path given. For example, using `/model_fb/controller/left` will return information for the left controller that is currently connected and will change if a different device that also represents a left controller is connected.

The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if `xrGetRenderModelPropertiesFB` is called with render model paths before calling `xrEnumerateRenderModelPathsFB`. The runtime **must** return



`XR_ERROR_PATH_INVALID` if a path not given by `xrEnumerateRenderModelPathsFB` is used.

If `xrGetRenderModelPropertiesFB` returns a success code of `XR_RENDER_MODEL_UNAVAILABLE_FB` and has a `XrRenderModelPropertiesFB::modelKey` of `XR_NULL_RENDER_MODEL_KEY_FB`, this indicates that the model for the device is unavailable. The application **may** keep calling `xrGetRenderModelPropertiesFB` because the model **may** become available later when a device is connected.

### Valid Usage (Implicit)

- The `XR_FB_render_model` extension **must** be enabled prior to calling `xrGetRenderModelPropertiesFB`
- `session` **must** be a valid `XrSession` handle
- `properties` **must** be a pointer to an `XrRenderModelPropertiesFB` structure

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_RENDER_MODEL_UNAVAILABLE_FB`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_CALL_ORDER_INVALID`

The `xrLoadRenderModelFB` function is defined as:

```
// Provided by XR_FB_render_model
XrResult xrLoadRenderModelFB(
    XrSession session,
    const XrRenderModelLoadInfoFB* info,
    XrRenderModelBufferFB* buffer);
```

## Parameter Descriptions

- `session` is the specified `XrSession`.
- `info` is a pointer to the `XrRenderModelLoadInfoFB` structure.
- `buffer` is a pointer to the `XrRenderModelBufferFB` structure to write the binary data into.

`xrLoadRenderModelFB` is used to load the GLTF model data using a valid `XrRenderModelLoadInfoFB::modelKey`. `xrLoadRenderModelFB` loads the model as a byte buffer containing the GLTF in the binary format (GLB). The GLB data **must** conform to the glTF 2.0 format defined at <https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html>. The GLB **may** contain texture data in a format that requires the use of the `KHR_texture_basisu` GLTF extension defined at [https://github.com/KhronosGroup/glTF/tree/main/extensions/2.0/Khronos/KHR\\_texture\\_basisu](https://github.com/KhronosGroup/glTF/tree/main/extensions/2.0/Khronos/KHR_texture_basisu). Therefore, the application **should** ensure it can handle this extension.

If the device for the requested model is disconnected or does not match the `XrRenderModelLoadInfoFB::modelKey` provided, `xrLoadRenderModelFB` **must** return `XR_RENDER_MODEL_UNAVAILABLE_FB` as well as an `XrRenderModelBufferFB::bufferCountOutput` value of 0 indicating that the model was not available.

The `xrLoadRenderModelFB` function **may** be slow, therefore applications **should** call it from a non-time sensitive thread.

## Valid Usage (Implicit)

- The `XR_FB_render_model` extension **must** be enabled prior to calling `xrLoadRenderModelFB`
- `session` **must** be a valid `XrSession` handle
- `info` **must** be a pointer to a valid `XrRenderModelLoadInfoFB` structure
- `buffer` **must** be a pointer to an `XrRenderModelBufferFB` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_RENDER_MODEL_UNAVAILABLE_FB`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_RENDER_MODEL_KEY_INVALID_FB`

## Issues

### Version History

- Revision 1, 2021-08-17 (Leonard Tsai)
  - Initial extension description
- Revision 2, 2022-05-03 (Robert Memmott)
  - Render Model Support Subsets
- Revision 3, 2022-07-07 (Rylie Pavlik, Collabora, Ltd.)
  - Fix implicit valid usage for `XrRenderModelCapabilitiesRequestFB`
- Revision 4, 2023-04-14 (Peter Chan)
  - Add possible render model path for `XR_META_virtual_keyboard`

## 12.66. XR\_FB\_scene

### Name String

`XR_FB_scene`

### Extension Type

Instance extension

## Registered Extension Number

176

## Revision

4

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_spatial\\_entity](#)

## Contributors

John Schofield, Facebook

Andrew Kim, Facebook

Yuichi Taguchi, Facebook

Cass Everitt, Facebook

## Overview

This extension expands on the concept of spatial entities to include a way for a spatial entity to represent rooms, objects, or other boundaries in a scene.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

```
// Provided by XR_FB_scene
typedef XrFlags64 XrSemanticLabelsSupportFlagsFB;
```

```

// Provided by XR_FB_scene
// Flag bits for XrSemanticLabelsSupportFlagsFB
static const XrSemanticLabelsSupportFlagsFB
XR_SEMANTIC_LABELS_SUPPORT_MULTIPLE_SEMANTIC_LABELS_BIT_FB = 0x00000001;
static const XrSemanticLabelsSupportFlagsFB
XR_SEMANTIC_LABELS_SUPPORT_ACCEPT_DESK_TO_TABLE_MIGRATION_BIT_FB = 0x00000002;
static const XrSemanticLabelsSupportFlagsFB
XR_SEMANTIC_LABELS_SUPPORT_ACCEPT_INVISIBLE_WALL_FACE_BIT_FB = 0x00000004;

```

## Flag Descriptions

- `XR_SEMANTIC_LABELS_SUPPORT_MULTIPLE_SEMANTIC_LABELS_BIT_FB` — If set, and the runtime reports the `extensionVersion` as 2 or greater, the runtime **may** return multiple semantic labels separated by a comma without spaces. Otherwise, the runtime **must** return a single semantic label.
- `XR_SEMANTIC_LABELS_SUPPORT_ACCEPT_DESK_TO_TABLE_MIGRATION_BIT_FB` — If set, and the runtime reports the `extensionVersion` as 3 or greater, the runtime **must** return "TABLE" instead of "DESK" as a semantic label to the application. Otherwise, the runtime **must** return "DESK" instead of "TABLE" as a semantic label to the application, when applicable.
- `XR_SEMANTIC_LABELS_SUPPORT_ACCEPT_INVISIBLE_WALL_FACE_BIT_FB` — If set, and the runtime reports the `extensionVersion` as 4 or greater, the runtime **may** return "INVISIBLE\_WALL\_FACE" instead of "WALL\_FACE" as a semantic label to the application in order to represent an invisible wall used to conceptually separate a space (e.g., separate a living space from a kitchen space in an open floor plan house even though there is no real wall between the two spaces) instead of a real wall. Otherwise, the runtime **must** return "WALL\_FACE" as a semantic label to the application in order to represent both an invisible and real wall, when applicable.

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SEMANTIC_LABELS_FB`
- `XR_TYPE_ROOM_LAYOUT_FB`
- `XR_TYPE_BOUNDARY_2D_FB`
- `XR_TYPE_SEMANTIC_LABELS_SUPPORT_INFO_FB`

## New Enums

## New Structures

The `XrExtent3DfFB` structure is defined as:

```

// Provided by XR_FB_scene
// XrExtent3DfFB is an alias for XrExtent3Df
typedef struct XrExtent3Df {
    float    width;
    float    height;
    float    depth;
} XrExtent3Df;

typedef XrExtent3Df XrExtent3DfFB;

```

## Member Descriptions

- **width** is the floating-point width of the extent.
- **height** is the floating-point height of the extent.
- **depth** is the floating-point depth of the extent.

This structure is used for component values that may be fractional (floating-point). If used to represent physical distances, values must be in meters. The width, height, and depth values must be non-negative.

## Valid Usage (Implicit)

- The [XR\\_FB\\_scene](#) extension **must** be enabled prior to using [XrExtent3DfFB](#)

The [XrOffset3DfFB](#) structure is defined as:

```

// Provided by XR_FB_scene
typedef struct XrOffset3DfFB {
    float    x;
    float    y;
    float    z;
} XrOffset3DfFB;

```

## Member Descriptions

- **x** is the floating-point offset in the x direction.
- **y** is the floating-point offset in the y direction.
- **z** is the floating-point offset in the z direction.

This structure is used for component values that may be fractional (floating-point). If used to represent physical distances, values must be in meters.

## Valid Usage (Implicit)

- The [XR\\_FB\\_scene](#) extension **must** be enabled prior to using [XrOffset3DfFB](#)

The [XrRect3DfFB](#) structure is defined as:

```
// Provided by XR_FB_scene
typedef struct XrRect3DfFB {
    XrOffset3DfFB    offset;
    XrExtent3DfFB   extent;
} XrRect3DfFB;
```

## Member Descriptions

- **offset** is the [XrOffset3DfFB](#) specifying the rectangle offset.
- **extent** is the [XrExtent3DfFB](#) specifying the rectangle extent.

This structure is used for component values that may be fractional (floating-point).

The bounding box is defined by an **offset** and **extent**. The **offset** refers to the coordinate of the minimum corner of the box in the local space of the [XrSpace](#); that is, the corner whose coordinate has the minimum value on each axis. The **extent** refers to the dimensions of the box along each axis. The maximum corner can therefore be computed as **offset** **extent**.

## Valid Usage (Implicit)

- The [XR\\_FB\\_scene](#) extension **must** be enabled prior to using [XrRect3DfFB](#)

The [XrSemanticLabelsFB](#) structure is defined as:

```
// Provided by XR_FB_scene
typedef struct XrSemanticLabelsFB {
    XrStructureType    type;
    const void*        next;
    uint32_t           bufferCapacityInput;
    uint32_t           bufferCountOutput;
    char*              buffer;
} XrSemanticLabelsFB;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain, such as [XrSemanticLabelsSupportInfoFB](#).
- `bufferCapacityInput` is the capacity of the `buffer` array, in bytes, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is the count of bytes written, or the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an array of bytes, but can be `NULL` if `bufferCapacityInput` is 0. Multiple labels represented by raw string, separated by a comma without spaces.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

This structure is used by the [xrGetSpaceSemanticLabelsFB](#) function to provide the application with the intended usage of the spatial entity.

### Valid Usage (Implicit)

- The [XR\\_FB\\_scene](#) extension **must** be enabled prior to using [XrSemanticLabelsFB](#)
- `type` **must** be `XR_TYPE_SEMANTIC_LABELS_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

The [XrRoomLayoutFB](#) structure is defined as:



```

// Provided by XR_FB_scene
typedef struct XrRoomLayoutFB {
    XrStructureType    type;
    const void*        next;
    XrUuidEXT          floorUuid;
    XrUuidEXT          ceilingUuid;
    uint32_t           wallUuidCapacityInput;
    uint32_t           wallUuidCountOutput;
    XrUuidEXT*         wallUuids;
} XrRoomLayoutFB;

```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `floorUuid` is the UUID of the spatial entity representing the room floor
- `ceilingUuid` is the UUID of the spatial entity representing the room ceiling
- `wallUuidCapacityInput` is the capacity of the `wallUuids` array, in number of UUIDs, or 0 to indicate a request to retrieve the required capacity.
- `wallUuidCountOutput` is the count of [XrUuidEXT](#) handles written, or the required capacity in the case that `wallUuidCapacityInput` is insufficient.
- `wallUuids` is a pointer to an array of [XrUuidEXT](#) handles, but can be `NULL` if `wallUuidCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `wallUuids` array size.

This structure is used by the [xrGetSpaceRoomLayoutFB](#) function to provide the application with the [XrUuidEXT](#) handles representing the various surfaces of a room.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to using `XrRoomLayoutFB`
- `type` **must** be `XR_TYPE_ROOM_LAYOUT_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- If `wallUuidCapacityInput` is not `0`, `wallUuids` **must** be a pointer to an array of `wallUuidCapacityInput` `XrUuidEXT` structures

The `XrBoundary2DFB` structure is defined as:

```
// Provided by XR_FB_scene
typedef struct XrBoundary2DFB {
    XrStructureType    type;
    const void*        next;
    uint32_t           vertexCapacityInput;
    uint32_t           vertexCountOutput;
    XrVector2f*        vertices;
} XrBoundary2DFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `vertexCapacityInput` is the capacity of the `vertices` array, in number of vertices, or `0` to indicate a request to retrieve the required capacity.
- `vertexCountOutput` is the count of `XrVector2f` written, or the required capacity in the case that `vertexCapacityInput` is insufficient.
- `vertices` is a pointer to an array of `XrVector2f`, but **can** be `NULL` if `vertexCapacityInput` is `0`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `vertices` array size.

This structure is used by the `xrGetSpaceBoundary2DFB` function to provide the application with the `XrVector2f` vertices representing the a spatial entity with a boundary.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to using `XrBoundary2DFB`
- `type` **must** be `XR_TYPE_BOUNDARY_2D_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- If `vertexCapacityInput` is not `0`, `vertices` **must** be a pointer to an array of `vertexCapacityInput` `XrVector2f` structures

The `XrSemanticLabelsSupportInfoFB` structure is defined as:

```
// Provided by XR_FB_scene
typedef struct XrSemanticLabelsSupportInfoFB {
    XrStructureType          type;
    const void*              next;
    XrSemanticLabelsSupportFlagsFB  flags;
    const char*              recognizedLabels;
} XrSemanticLabelsSupportInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of `XrSemanticLabelsSupportFlagBitsFB` that specifies additional behaviors.
- `recognizedLabels` is a `NULL` terminated string that indicates a set of semantic labels recognized by the application. Each semantic label **must** be represented as a string and be separated by a comma without spaces. This field **must** include at least "OTHER" and **must** not be `NULL`.

The `XrSemanticLabelsSupportInfoFB` structure **may** be specified in the `next` chain of `XrSemanticLabelsFB` to specify additional behaviors of the `xrGetSpaceSemanticLabelsFB` function. The runtime **must** follow the behaviors specified in `flags` according to the descriptions of `XrSemanticLabelsSupportFlagBitsFB`. The runtime **must** return any semantic label that is not included in `recognizedLabels` as "OTHER" to the application. The runtime **must** follow this direction only if the runtime reports the `XrExtensionProperties::extensionVersion` as 2 or greater, otherwise the runtime **must** ignore this as an unknown chained structure.

If the `XrSemanticLabelsSupportInfoFB` structure is not present in the `next` chain of `XrSemanticLabelsFB`, the runtime **may** return any semantic labels to the application.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to using `XrSemanticLabelsSupportInfoFB`
- `type` **must** be `XR_TYPE_SEMANTIC_LABELS_SUPPORT_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `flags` **must** be `0` or a valid combination of `XrSemanticLabelsSupportFlagBitsFB` values
- `recognizedLabels` **must** be a null-terminated UTF-8 string

## New Functions

The `xrGetSpaceBoundingBox2DFB` function is defined as:

```
// Provided by XR_FB_scene
XrResult xrGetSpaceBoundingBox2DFB(
    XrSession          session,
    XrSpace            space,
    XrRect2Df*        boundingBox2DOutput);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession`.
- `space` is the `XrSpace` handle to the spatial entity.
- `boundingBox2DOutput` is an output parameter pointing to the structure containing the 2D bounding box for `space`.

Gets the 2D bounding box for a spatial entity with the `XR_SPACE_COMPONENT_TYPE_BOUNDED_2D_FB` component type enabled.

The bounding box is defined by an `XrRect2Df::offset` and `XrRect2Df::extent`. The `XrRect2Df::offset` refers to the coordinate of the minimum corner of the box in the x-y plane of the given `XrSpace`'s coordinate system; that is, the corner whose coordinate has the minimum value on each axis. The `XrRect2Df::extent` refers to the dimensions of the box along each axis. The maximum corner can therefore be computed as `XrRect2Df::offset` `XrRect2Df::extent`.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to calling `xrGetSpaceBoundingBox2DFB`
- `session` **must** be a valid `XrSession` handle
- `space` **must** be a valid `XrSpace` handle
- `boundingBox2DOutput` **must** be a pointer to an `XrRect2Df` structure
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetSpaceBoundingBox3DFB` function is defined as:

```
// Provided by XR_FB_scene
XrResult xrGetSpaceBoundingBox3DFB(
    XrSession          session,
    XrSpace            space,
    XrRect3DfFB*      boundingBox3DOutput);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `space` is the [XrSpace](#) handle to the spatial entity.
- `boundingBox3DOutput` is an output parameter pointing to the structure containing the 3D bounding box for `space`.

Gets the 3D bounding box for a spatial entity with the `XR_SPACE_COMPONENT_TYPE_BOUNDED_3D_FB` component type enabled.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to calling `xrGetSpaceBoundingBox3DFB`
- `session` **must** be a valid [XrSession](#) handle
- `space` **must** be a valid [XrSpace](#) handle
- `boundingBox3DOutput` **must** be a pointer to an [XrRect3DfFB](#) structure
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetSpaceSemanticLabelsFB` function is defined as:

```
// Provided by XR_FB_scene
XrResult xrGetSpaceSemanticLabelsFB(
    XrSession          session,
    XrSpace            space,
    XrSemanticLabelsFB* semanticLabelsOutput);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `space` is the [XrSpace](#) handle to the spatial entity.
- `semanticLabelsOutput` is an output parameter pointing to the structure containing the [XrSemanticLabelsFB](#) for `space`.

Gets the semantic labels for a spatial entity with the `XR_SPACE_COMPONENT_TYPE_SEMANTIC_LABELS_FB` component type enabled.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to calling `xrGetSpaceSemanticLabelsFB`
- `session` **must** be a valid [XrSession](#) handle
- `space` **must** be a valid [XrSpace](#) handle
- `semanticLabelsOutput` **must** be a pointer to an [XrSemanticLabelsFB](#) structure
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetSpaceBoundary2DFB` function is defined as:

```
// Provided by XR_FB_scene
XrResult xrGetSpaceBoundary2DFB(
    XrSession          session,
    XrSpace            space,
    XrBoundary2DFB*   boundary2DOutput);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession`.
- `space` is the `XrSpace` handle to the spatial entity.
- `boundary2DOutput` is an output parameter pointing to the structure containing the `XrBoundary2DFB` for `space`.

Gets the 2D boundary, specified by vertices, for a spatial entity with the `XR_SPACE_COMPONENT_TYPE_BOUNDED_2D_FB` component type enabled.



## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to calling `xrGetSpaceBoundary2DFB`
- `session` **must** be a valid `XrSession` handle
- `space` **must** be a valid `XrSpace` handle
- `boundary2DOutput` **must** be a pointer to an `XrBoundary2DFB` structure
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetSpaceRoomLayoutFB` function is defined as:

```
// Provided by XR_FB_scene
XrResult xrGetSpaceRoomLayoutFB(
    XrSession          session,
    XrSpace            space,
    XrRoomLayoutFB*   roomLayoutOutput);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession`.
- `space` is the `XrSpace` handle to the spatial entity.
- `roomLayoutOutput` is an output parameter pointing to the structure containing the `XrRoomLayoutFB` for `space`.

Gets the room layout, specified by UUIDs for each surface, for a spatial entity with the `XR_SPACE_COMPONENT_TYPE_ROOM_LAYOUT_FB` component type enabled.

If the `XrRoomLayoutFB::wallUuidCapacityInput` field is zero (indicating a request to retrieve the required capacity for the `XrRoomLayoutFB::wallUuids` array), or if `xrGetSpaceRoomLayoutFB` returns failure, then the values of `floorUuid` and `ceilingUuid` are unspecified and should not be used.

## Valid Usage (Implicit)

- The `XR_FB_scene` extension **must** be enabled prior to calling `xrGetSpaceRoomLayoutFB`
- `session` **must** be a valid `XrSession` handle
- `space` **must** be a valid `XrSpace` handle
- `roomLayoutOutput` **must** be a pointer to an `XrRoomLayoutFB` structure
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

### Version History

- Revision 1, 2022-03-09 (John Schofield)
  - Initial draft
- Revision 2, 2023-04-03 (Yuichi Taguchi)
  - Introduce [XrSemanticLabelsSupportInfoFB](#).
- Revision 3, 2023-04-03 (Yuichi Taguchi)
  - Introduce `XR_SEMANTIC_LABELS_SUPPORT_ACCEPT_DESK_TO_TABLE_MIGRATION_BIT_FB`.
- Revision 4, 2023-06-12 (Yuichi Taguchi)
  - Introduce `XR_SEMANTIC_LABELS_SUPPORT_ACCEPT_INVISIBLE_WALL_FACE_BIT_FB`.

## 12.67. XR\_FB\_scene\_capture

### Name String

`XR_FB_scene_capture`

### Extension Type

Instance extension

## Registered Extension Number

199

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

John Schofield, Facebook

Andrew Kim, Facebook

Yuichi Taguchi, Facebook

Cass Everitt, Facebook

## Overview

This extension allows an application to request that the system begin capturing information about what is in the environment around the user.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SCENE\\_CAPTURE\\_REQUEST\\_INFO\\_FB](#)
- [XR\\_TYPE\\_EVENT\\_DATA\\_SCENE\\_CAPTURE\\_COMPLETE\\_FB](#)

## New Enums

## New Structures

The [XrSceneCaptureRequestInfoFB](#) structure is defined as:

```
// Provided by XR_FB_scene_capture
typedef struct XrSceneCaptureRequestInfoFB {
    XrStructureType    type;
    const void*        next;
    uint32_t           requestByteCount;
    const char*        request;
} XrSceneCaptureRequestInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestByteCount` is byte length of the `request` parameter.
- `request` is a string which the application **can** use to specify which type of scene capture should be initiated by the runtime. The contents of buffer pointed to by the `request` parameter is runtime-specific.

The [XrSceneCaptureRequestInfoFB](#) structure is used by an application to instruct the system what to look for during a scene capture. If the `request` parameter is `NULL`, then the runtime **must** conduct a default scene capture.

## Valid Usage (Implicit)

- The [XR\\_FB\\_scene\\_capture](#) extension **must** be enabled prior to using [XrSceneCaptureRequestInfoFB](#)
- `type` **must** be `XR_TYPE_SCENE_CAPTURE_REQUEST_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `requestByteCount` is not `0`, `request` **must** be a pointer to an array of `requestByteCount` char values

The [XrEventDataSceneCaptureCompleteFB](#) structure is defined as:

```
// Provided by XR_FB_scene_capture
typedef struct XrEventDataSceneCaptureCompleteFB {
    XrStructureType    type;
    const void*        next;
    XrAsyncRequestIdFB requestId;
    XrResult            result;
} XrEventDataSceneCaptureCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous query request.
- `result` is an [XrResult](#) that indicates if the request succeeded or if an error occurred.

The [XrEventDataSceneCaptureCompleteFB](#) structure is used by an application to instruct the system what to look for during a scene capture.

## Valid Usage (Implicit)

- The [XR\\_FB\\_scene\\_capture](#) extension **must** be enabled prior to using [XrEventDataSceneCaptureCompleteFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_SCENE_CAPTURE_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `result` **must** be a valid [XrResult](#) value

## New Functions

The [xrRequestSceneCaptureFB](#) function is defined as:

```
// Provided by XR_FB_scene_capture
XrResult xrRequestSceneCaptureFB(
    XrSession                session,
    const XrSceneCaptureRequestInfoFB* info,
    XrAsyncRequestIdFB*      requestId);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `info` is an [XrSceneCaptureRequestInfoFB](#) which specifies how the scene capture should occur.
- `requestId` is the output parameter that points to the ID of this asynchronous request.

The [xrRequestSceneCaptureFB](#) function is used by an application to begin capturing the scene around the user. This is an asynchronous operation.

## Valid Usage (Implicit)

- The [XR\\_FB\\_scene\\_capture](#) extension **must** be enabled prior to calling [xrRequestSceneCaptureFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `info` **must** be a pointer to a valid [XrSceneCaptureRequestInfoFB](#) structure
- `requestId` **must** be a pointer to an [XrAsyncRequestIdFB](#) value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

## Version History

- Revision 1, 2022-03-09 (John Schofield)
  - Initial draft

## 12.68. XR\_FB\_space\_warp

### Name String

`XR_FB_space_warp`

### Extension Type

Instance extension

### Registered Extension Number

172

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Jian Zhang, Facebook

Neel Bedekar, Facebook

Xiang Wei, Facebook

### Overview

This extension provides support to enable space warp technology on application. By feeding application generated motion vector and depth buffer images, the runtime can do high quality frame extrapolation and reprojection, allow applications to run at half fps but still providing smooth experience to users.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

#### Note



This extension is independent of `XR_KHR_composition_layer_depth`, and both may be enabled and used at the same time, for different purposes. The `XrCompositionLayerSpaceWarpInfoFB::depthSubImage` depth data is dedicated for space warp, and its resolution is usually lower than `XrCompositionLayerDepthInfoKHR::subImage`. See `XrSystemSpaceWarpPropertiesFB` for suggested resolution of `depthSubImage`.



## New Flag Types

```
typedef XrFlags64 XrCompositionLayerSpaceWarpInfoFlagsFB;
```

```
// Flag bits for XrCompositionLayerSpaceWarpInfoFlagsFB
static const XrCompositionLayerSpaceWarpInfoFlagsFB
XR_COMPOSITION_LAYER_SPACE_WARP_INFO_FRAME_SKIP_BIT_FB = 0x00000001;
```

### Flag Descriptions

- `XR_COMPOSITION_LAYER_SPACE_WARP_INFO_FRAME_SKIP_BIT_FB` requests that the runtime skips space warp frame extrapolation for a particular frame. This can be used when the application has better knowledge the particular frame will be not a good fit for space warp frame extrapolation.

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_SPACE_WARP_INFO_FB`
- `XR_TYPE_SYSTEM_SPACE_WARP_PROPERTIES_FB`

## New Enums

- `XR_COMPOSITION_LAYER_SPACE_WARP_INFO_FRAME_SKIP_BIT_FB`

## New Structures

When submitting motion vector buffer and depth buffers along with projection layers, add an `XrCompositionLayerSpaceWarpInfoFB` structure to the `XrCompositionLayerProjectionView::next` chain, for each `XrCompositionLayerProjectionView` structure in the given layer.

The `XrCompositionLayerSpaceWarpInfoFB` structure is defined as:

```
// Provided by XR_FB_space_warp
typedef struct XrCompositionLayerSpaceWarpInfoFB {
    XrStructureType                type;
    const void*                    next;
    XrCompositionLayerSpaceWarpInfoFlagsFB layerFlags;
    XrSwapchainSubImage            motionVectorSubImage;
    XrPosef                        appSpaceDeltaPose;
    XrSwapchainSubImage            depthSubImage;
    float                          minDepth;
    float                          maxDepth;
    float                          nearZ;
    float                          farZ;
} XrCompositionLayerSpaceWarpInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `layerFlags` is a bitmask of [XrCompositionLayerSpaceWarpInfoFlagsFB](#).
- `motionVectorSubImage` identifies the motion vector image [XrSwapchainSubImage](#) to be associated with the submitted layer [XrCompositionLayerProjection](#).
- `appSpaceDeltaPose` is the incremental application-applied transform, if any, since the previous frame that affects the view. When artificial locomotion (scripted movement, teleportation, etc.) happens, the application might transform the whole [XrCompositionLayerProjection::space](#) from one application space pose to another pose between frames. The pose should be identity when there is no [XrCompositionLayerProjection::space](#) transformation in application.
- `depthSubImage` identifies the depth image [XrSwapchainSubImage](#) to be associated with `motionVectorSubImage`. The swapchain should be created with `XR_SWAPCHAIN_USAGE_SAMPLED_BIT | XR_SWAPCHAIN_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`.
- `minDepth` and `maxDepth` are the range of depth values the depth swapchain could have, in the range of [0.0,1.0]. This is akin to min and max values of OpenGL's `glDepthRange`, but with the requirement here that `maxDepth`  $\geq$  `minDepth`.
- `nearZ` is the positive distance in meters of the `minDepth` value in the depth swapchain. Applications **may** use a `nearZ` that is greater than `farZ` to indicate depth values are reversed. `nearZ` can be infinite.
- `farZ` is the positive distance in meters of the `maxDepth` value in the depth swapchain. `farZ` can be infinite.

The motion vector data is stored in the `motionVectorSubImage`'s RGB channels, defined in NDC

(normalized device coordinates) space, for example, the same surface point's NDC is PrevNDC in previous frame, CurrNDC in current frame, then the motion vector value is "highp vec3 motionVector = ( CurrNDC - PrevNDC ).xyz;". Signed 16 bit float pixel format is recommended for this image.

The runtime **must** return error `XR_ERROR_VALIDATION_FAILURE` if `nearZ == farZ`.

### Valid Usage (Implicit)

- The `XR_FB_space_warp` extension **must** be enabled prior to using `XrCompositionLayerSpaceWarpInfoFB`
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_SPACE_WARP_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layerFlags` **must** be `0` or a valid combination of `XrCompositionLayerSpaceWarpInfoFlagBitsFB` values
- `motionVectorSubImage` **must** be a valid `XrSwapchainSubImage` structure
- `depthSubImage` **must** be a valid `XrSwapchainSubImage` structure

When this extension is enabled, an application **can** pass in an `XrSystemSpaceWarpPropertiesFB` structure in the `XrSystemProperties::next` chain when calling `xrGetSystemProperties` to acquire information about recommended motion vector buffer resolution. The `XrSystemSpaceWarpPropertiesFB` structure is defined as:

```
// Provided by XR_FB_space_warp
typedef struct XrSystemSpaceWarpPropertiesFB {
    XrStructureType    type;
    void*              next;
    uint32_t           recommendedMotionVectorImageRectWidth;
    uint32_t           recommendedMotionVectorImageRectHeight;
} XrSystemSpaceWarpPropertiesFB;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `recommendedMotionVectorImageRectWidth`: recommended motion vector and depth image width
- `recommendedMotionVectorImageRectHeight`: recommended motion vector and depth image height

## Valid Usage (Implicit)

- The `XR_FB_space_warp` extension **must** be enabled prior to using `XrSystemSpaceWarpPropertiesFB`
- `type` **must** be `XR_TYPE_SYSTEM_SPACE_WARP_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## Issues

## Version History

- Revision 1, 2021-08-04 (Jian Zhang)
  - Initial extension description
- Revision 2, 2022-02-07 (Jian Zhang)
  - Add `XR_COMPOSITION_LAYER_SPACE_WARP_INFO_FRAME_SKIP_BIT_FB`

# 12.69. XR\_FB\_spatial\_entity

## Name String

`XR_FB_spatial_entity`

## Extension Type

Instance extension

## Registered Extension Number

114

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

John Schofield, Facebook  
Andrew Kim, Facebook  
Yuichi Taguchi, Facebook  
Cass Everitt, Facebook  
Curtis Arink, Facebook

## Overview

This extension enables applications to use spatial entities to specify world-locked frames of reference. It enables applications to persist the real world location of content over time and contains definitions for the Entity-Component System. All Facebook spatial entity and scene extensions are dependent on this one.

We use OpenXR [XrSpace](#) handles to give applications access to spatial entities such as Spatial Anchors. In other words, any operation which involves spatial entities uses [XrSpace](#) handles to identify the affected spatial entities.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

This extension allows:

- An application to create a Spatial Anchor (a type of spatial entity).
- An application to enumerate supported components for a given spatial entity.
- An application to enable or disable a component for a given spatial entity.
- An application to get the status of a component for a given spatial entity.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SYSTEM\\_SPATIAL\\_ENTITY\\_PROPERTIES\\_FB](#)
- [XR\\_TYPE\\_SPATIAL\\_ANCHOR\\_CREATE\\_INFO\\_FB](#)
- [XR\\_TYPE\\_SPACE\\_COMPONENT\\_STATUS\\_SET\\_INFO\\_FB](#)
- [XR\\_TYPE\\_SPACE\\_COMPONENT\\_STATUS\\_FB](#)
- [XR\\_TYPE\\_EVENT\\_DATA\\_SPATIAL\\_ANCHOR\\_CREATE\\_COMPLETE\\_FB](#)
- [XR\\_TYPE\\_EVENT\\_DATA\\_SPACE\\_SET\\_STATUS\\_COMPLETE\\_FB](#)

[XrResult](#) enumeration is extended with:

- [XR\\_ERROR\\_SPACE\\_COMPONENT\\_NOT\\_SUPPORTED\\_FB](#)
- [XR\\_ERROR\\_SPACE\\_COMPONENT\\_NOT\\_ENABLED\\_FB](#)
- [XR\\_ERROR\\_SPACE\\_COMPONENT\\_STATUS\\_PENDING\\_FB](#)
- [XR\\_ERROR\\_SPACE\\_COMPONENT\\_STATUS\\_ALREADY\\_SET\\_FB](#)

## New Enums

```

// Provided by XR_FB_spatial_entity
typedef enum XrSpaceComponentTypeFB {
    XR_SPACE_COMPONENT_TYPE_LOCATABLE_FB = 0,
    XR_SPACE_COMPONENT_TYPE_STORABLE_FB = 1,
    XR_SPACE_COMPONENT_TYPE_SHARABLE_FB = 2,
    XR_SPACE_COMPONENT_TYPE_BOUNDED_2D_FB = 3,
    XR_SPACE_COMPONENT_TYPE_BOUNDED_3D_FB = 4,
    XR_SPACE_COMPONENT_TYPE_SEMANTIC_LABELS_FB = 5,
    XR_SPACE_COMPONENT_TYPE_ROOM_LAYOUT_FB = 6,
    XR_SPACE_COMPONENT_TYPE_SPACE_CONTAINER_FB = 7,
// Provided by XR_META_spatial_entity_mesh
    XR_SPACE_COMPONENT_TYPE_TRIANGLE_MESH_META = 1000269000,
    XR_SPACE_COMPONENT_TYPE_MAX_ENUM_FB = 0x7FFFFFFF
} XrSpaceComponentTypeFB;

```

Specify the component interfaces attached to the spatial entity.

### Enumerant Descriptions

- **XR\_SPACE\_COMPONENT\_TYPE\_LOCATABLE\_FB** — Enables tracking the 6 DOF pose of the [XrSpace](#) with [xrLocateSpace](#).
- **XR\_SPACE\_COMPONENT\_TYPE\_STORABLE\_FB** — Enables persistence operations: save and erase.
- **XR\_SPACE\_COMPONENT\_TYPE\_SHARABLE\_FB** — Enables sharing of spatial entities.
- **XR\_SPACE\_COMPONENT\_TYPE\_BOUNDED\_2D\_FB** — Bounded 2D component.
- **XR\_SPACE\_COMPONENT\_TYPE\_BOUNDED\_3D\_FB** — Bounded 3D component.
- **XR\_SPACE\_COMPONENT\_TYPE\_SEMANTIC\_LABELS\_FB** — Semantic labels component.
- **XR\_SPACE\_COMPONENT\_TYPE\_ROOM\_LAYOUT\_FB** — Room layout component.
- **XR\_SPACE\_COMPONENT\_TYPE\_SPACE\_CONTAINER\_FB** — Space container component.

### New Base Types

The [XrAsyncRequestIdFB](#) base type is defined as:

```

// Provided by XR_FB_spatial_entity
XR_DEFINE_ATOM(XrAsyncRequestIdFB)

```

Represents a request to the spatial entity system. Several functions in this and other extensions will

populate an output variable of this type so that an application **can** use it when referring to a specific request.

## New Structures

The `XrSystemSpatialEntityPropertiesFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity
typedef struct XrSystemSpatialEntityPropertiesFB {
    XrStructureType    type;
    const void*        next;
    XrBool32           supportsSpatialEntity;
} XrSystemSpatialEntityPropertiesFB;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `supportsSpatialEntity` is a boolean value that determines if spatial entities are supported by the system.

An application **can** inspect whether the system is capable of spatial entity operations by extending the `XrSystemProperties` with `XrSystemSpatialEntityPropertiesFB` structure when calling `xrGetSystemProperties`.

If a runtime returns `XR_FALSE` for `supportsSpatialEntity`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrGetSpaceUuidFB`.

### Valid Usage (Implicit)

- The `XR_FB_spatial_entity` extension **must** be enabled prior to using `XrSystemSpatialEntityPropertiesFB`
- `type` **must** be `XR_TYPE_SYSTEM_SPATIAL_ENTITY_PROPERTIES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrSpatialAnchorCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity
typedef struct XrSpatialAnchorCreateInfoFB {
    XrStructureType    type;
    const void*        next;
    XrSpace             space;
    XrPosef            poseInSpace;
    XrTime             time;
} XrSpatialAnchorCreateInfoFB;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **space** is the [XrSpace](#) handle to the reference space that defines the **poseInSpace** of the anchor to be defined.
- **poseInSpace** is the [XrPosef](#) location and orientation of the Spatial Anchor in the specified reference space.
- **time** is the [XrTime](#) timestamp associated with the specified pose.

Parameters to create a new spatial anchor.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity](#) extension **must** be enabled prior to using [XrSpatialAnchorCreateInfoFB](#)
- **type** **must** be `XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_FB`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **space** **must** be a valid [XrSpace](#) handle

The [XrSpaceComponentStatusSetInfoFB](#) structure is defined as:



```
// Provided by XR_FB_spatial_entity
typedef struct XrSpaceComponentStatusSetInfoFB {
    XrStructureType      type;
    const void*          next;
    XrSpaceComponentTypeFB componentType;
    XrBool32             enabled;
    XrDuration           timeout;
} XrSpaceComponentStatusSetInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `componentType` is the component whose status is to be set.
- `enabled` is the value to set the component to.
- `timeout` is the number of nanoseconds before the operation should be cancelled. A value of [XR\\_INFINITE\\_DURATION](#) indicates to never time out. See [Duration](#) for more details.

Enables or disables the specified component for the specified spatial entity.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity](#) extension **must** be enabled prior to using [XrSpaceComponentStatusSetInfoFB](#)
- `type` **must** be `XR_TYPE_SPACE_COMPONENT_STATUS_SET_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `componentType` **must** be a valid [XrSpaceComponentTypeFB](#) value

The [XrSpaceComponentStatusFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity
typedef struct XrSpaceComponentStatusFB {
    XrStructureType    type;
    void*              next;
    XrBool32           enabled;
    XrBool32           changePending;
} XrSpaceComponentStatusFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `enabled` is a boolean value that determines if a component is currently enabled or disabled.
- `changePending` is a boolean value that determines if the component's enabled state is about to change.

It holds information on the current state of a component.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity` extension **must** be enabled prior to using `XrSpaceComponentStatusFB`
- `type` **must** be `XR_TYPE_SPACE_COMPONENT_STATUS_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrEventDataSpatialAnchorCreateCompleteFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity
typedef struct XrEventDataSpatialAnchorCreateCompleteFB {
    XrStructureType    type;
    const void*        next;
    XrAsyncRequestIdFB requestId;
    XrResult            result;
    XrSpace             space;
    XrUuidEXT           uuid;
} XrEventDataSpatialAnchorCreateCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous request used to create a new spatial anchor.
- `result` is an [XrResult](#) that determines if the request succeeded or if an error occurred.
- `space` is the [XrSpace](#) handle to the newly created spatial anchor.
- `uuid` is the UUID of the newly created spatial anchor.

It describes the result of a request to create a new spatial anchor. Once this event is posted, it is the applications responsibility to take ownership of the [XrSpace](#). The [XrSession](#) passed into [xrCreateSpatialAnchorFB](#) is the parent handle of the newly created [XrSpace](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity](#) extension **must** be enabled prior to using [XrEventDataSpatialAnchorCreateCompleteFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_SPATIAL_ANCHOR_CREATE_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataSpaceSetStatusCompleteFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity
typedef struct XrEventDataSpaceSetStatusCompleteFB {
    XrStructureType      type;
    const void*          next;
    XrAsyncRequestIdFB   requestId;
    XrResult              result;
    XrSpace               space;
    XrUuidEXT             uuid;
    XrSpaceComponentTypeFB componentType;
    XrBool32              enabled;
} XrEventDataSpaceSetStatusCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous request used to enable or disable a component.
- `result` is an [XrResult](#) that describes whether the request succeeded or if an error occurred.
- `space` is the [XrSpace](#) handle to the spatial entity.
- `uuid` is the UUID of the spatial entity.
- `componentType` is the type of component being enabled or disabled.
- `enabled` is a boolean value indicating whether the component is now enabled or disabled.

It describes the result of a request to enable or disable a component of a spatial entity.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity](#) extension **must** be enabled prior to using [XrEventDataSpaceSetStatusCompleteFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_SPACE_SET_STATUS_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrCreateSpatialAnchorFB](#) function is defined as:

```
// Provided by XR_FB_spatial_entity
XrResult xrCreateSpatialAnchorFB(
    XrSession session,
    const XrSpatialAnchorCreateInfoFB* info,
    XrAsyncRequestIdFB* requestId);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `info` is a pointer to an [XrSpatialAnchorCreateInfoFB](#) structure containing information about how to create the anchor.
- `requestId` is the output parameter that points to the ID of this asynchronous request.

Creates a Spatial Anchor using the specified tracking origin and pose relative to the specified tracking origin. The anchor will be locatable at the time of creation, and the 6 DOF pose relative to the tracking origin **can** be queried using the [xrLocateSpace](#) method. This operation is asynchronous and the runtime **must** post an [XrEventDataSpatialAnchorCreateCompleteFB](#) event when the operation completes successfully or encounters an error. If this function returns a failure code, no event is posted. The `requestId` **can** be used to later refer to the request, such as identifying which request has completed when an [XrEventDataSpatialAnchorCreateCompleteFB](#) is posted to the event queue.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity` extension **must** be enabled prior to calling [xrCreateSpatialAnchorFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `info` **must** be a pointer to a valid [XrSpatialAnchorCreateInfoFB](#) structure
- `requestId` **must** be a pointer to an [XrAsyncRequestIdFB](#) value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_TIME_INVALID`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetSpaceUuidFB` function is defined as:

```
// Provided by XR_FB_spatial_entity
XrResult xrGetSpaceUuidFB(
    XrSpace          space,
    XrUuidEXT*      uuid);
```

## Parameter Descriptions

- `space` is the `XrSpace` handle of a spatial entity.
- `uuid` is an output parameter pointing to the entity's UUID.

Gets the UUID for a spatial entity. If this space was previously created as a spatial anchor, `uuid` **must** be equal to the `XrEventDataSpatialAnchorCreateCompleteFB::uuid` in the event corresponding to the creation of that space. Subsequent calls to `xrGetSpaceUuidFB` using the same `XrSpace` **must** return the same `XrUuidEXT`.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity` extension **must** be enabled prior to calling `xrGetSpaceUuidFB`
- `space` **must** be a valid `XrSpace` handle
- `uuid` **must** be a pointer to an `XrUuidEXT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrEnumerateSpaceSupportedComponentsFB` function is defined as:

```
// Provided by XR_FB_spatial_entity
XrResult xrEnumerateSpaceSupportedComponentsFB(
    XrSpace                                space,
    uint32_t                               componentTypeCapacityInput,
    uint32_t*                              componentTypeCountOutput,
    XrSpaceComponentTypeFB*               componentTypes);
```

## Parameter Descriptions

- `space` is the `XrSpace` handle to the spatial entity.
- `componentTypeCapacityInput` is the capacity of the `componentTypes` array, or 0 to indicate a request to retrieve the required capacity.
- `componentTypeCountOutput` is a pointer to the count of `componentTypes` written, or a pointer to the required capacity in the case that `componentTypeCapacityInput` is insufficient.
- `componentTypes` is a pointer to an array of `XrSpaceComponentTypeFB` values, but **can** be `NULL` if `componentTypeCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `componentTypes` size.

Lists any component types that an entity supports. The list of component types available for an entity depends on which extensions are enabled. Component types **must** not be enumerated unless the corresponding extension that defines them is also enabled.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity` extension **must** be enabled prior to calling `xrEnumerateSpaceSupportedComponentsFB`
- `space` **must** be a valid `XrSpace` handle
- `componentTypeCountOutput` **must** be a pointer to a `uint32_t` value
- If `componentTypeCapacityInput` is not 0, `componentTypes` **must** be a pointer to an array of `componentTypeCapacityInput` `XrSpaceComponentTypeFB` values



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrSetSpaceComponentStatusFB` function is defined as:

```
// Provided by XR_FB_spatial_entity
XrResult xrSetSpaceComponentStatusFB(
    XrSpace space,
    const XrSpaceComponentStatusSetInfoFB* info,
    XrAsyncRequestIdFB* requestId);
```

## Parameter Descriptions

- `space` is the `XrSpace` handle to the spatial entity.
- `info` is a pointer to an `XrSpaceComponentStatusSetInfoFB` structure containing information about the component to be enabled or disabled.
- `requestId` is the output parameter that points to the ID of this asynchronous request.

Enables or disables the specified component for the specified entity. This operation is asynchronous and always returns immediately, regardless of the value of `XrSpaceComponentStatusSetInfoFB::timeout`. The `requestId` can be used to later refer to the request, such as identifying which request has completed when an `XrEventDataSpaceSetStatusCompleteFB` is posted to the event queue. If this function returns a failure code, no event is posted. This function **must** return

`XR_ERROR_SPACE_COMPONENT_NOT_SUPPORTED_FB` if the `XrSpace` does not support the specified component type.

### Valid Usage (Implicit)

- The `XR_FB_spatial_entity` extension **must** be enabled prior to calling `xrSetSpaceComponentStatusFB`
- `space` **must** be a valid `XrSpace` handle
- `info` **must** be a pointer to a valid `XrSpaceComponentStatusSetInfoFB` structure
- `requestId` **must** be a pointer to an `XrAsyncRequestIdFB` value

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_COMPONENT_STATUS_PENDING_FB`
- `XR_ERROR_SPACE_COMPONENT_STATUS_ALREADY_SET_FB`
- `XR_ERROR_SPACE_COMPONENT_NOT_SUPPORTED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetSpaceComponentStatusFB` function is defined as:

```
// Provided by XR_FB_spatial_entity
XrResult xrGetSpaceComponentStatusFB(
    XrSpace space,
    XrSpaceComponentTypeFB componentType,
    XrSpaceComponentStatusFB* status);
```

## Parameter Descriptions

- **space** is the [XrSpace](#) handle of a spatial entity.
- **componentType** is the component type to query.
- **status** is an output parameter pointing to the structure containing the status of the component that was queried.

Gets the current status of the specified component for the specified entity. This function **must** return [XR\\_ERROR\\_SPACE\\_COMPONENT\\_NOT\\_SUPPORTED\\_FB](#) if the [XrSpace](#) does not support the specified component type.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity](#) extension **must** be enabled prior to calling [xrGetSpaceComponentStatusFB](#)
- **space** **must** be a valid [XrSpace](#) handle
- **componentType** **must** be a valid [XrSpaceComponentTypeFB](#) value
- **status** **must** be a pointer to an [XrSpaceComponentStatusFB](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_COMPONENT_NOT_SUPPORTED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

### Version History

- Revision 1, 2022-01-22 (John Schofield)
  - Initial draft
- Revision 2, 2023-01-18 (Andrew Kim)
  - Added a new component enum value
- Revision 3, 2023-01-30 (Wenlin Mao)
  - Drop requirement for `XR_EXT_uuid` must be enabled

## 12.70. XR\_FB\_spatial\_entity\_container

### Name String

`XR_FB_spatial_entity_container`

### Extension Type

Instance extension

### Registered Extension Number

200

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_spatial\\_entity](#)

## Contributors

John Schofield, Facebook

Andrew Kim, Facebook

Yuichi Taguchi, Facebook

## Overview

This extension expands on the concept of spatial entities to include a way for one spatial entity to contain multiple child spatial entities, forming a hierarchy.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SPACE\\_CONTAINER\\_FB](#)

## New Enums

## New Structures

The [XrSpaceContainerFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_container
typedef struct XrSpaceContainerFB {
    XrStructureType    type;
    const void*        next;
    uint32_t            uuidCapacityInput;
    uint32_t            uuidCountOutput;
    XrUuidEXT*         uuids;
} XrSpaceContainerFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `uuidCapacityInput` is the capacity of the `uuids` array, or 0 to indicate a request to retrieve the required capacity.
- `uuidCountOutput` is an output parameter which will hold the number of UUIDs included in the output list, or the required capacity in the case that `uuidCapacityInput` is insufficient
- `uuids` is an output parameter which will hold a list of space UUIDs contained by the space to which the component is attached.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `uuids` size.

The [XrSpaceContainerFB](#) structure **can** be used by an application to perform the two calls required to obtain information about which spatial entities are contained by a specified spatial entity.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_container](#) extension **must** be enabled prior to using [XrSpaceContainerFB](#)
- `type` **must** be `XR_TYPE_SPACE_CONTAINER_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `uuidCapacityInput` is not 0, `uuids` **must** be a pointer to an array of `uuidCapacityInput` [XrUuidEXT](#) structures

## New Functions

The [xrGetSpaceContainerFB](#) function is defined as:

```
// Provided by XR_FB_spatial_entity_container
XrResult xrGetSpaceContainerFB(
    XrSession          session,
    XrSpace            space,
    XrSpaceContainerFB* spaceContainerOutput);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `space` is a handle to an [XrSpace](#).
- `spaceContainerOutput` is the output parameter that points to an [XrSpaceContainerFB](#) containing information about which spaces are contained by `space`.

The [xrGetSpaceContainerFB](#) function is used by an application to perform the two calls required to obtain information about which spatial entities are contained by a specified spatial entity.

The `XR_SPACE_COMPONENT_TYPE_SPACE_CONTAINER_FB` component type **must** be enabled, otherwise this function will return `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_container](#) extension **must** be enabled prior to calling [xrGetSpaceContainerFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `space` **must** be a valid [XrSpace](#) handle
- `spaceContainerOutput` **must** be a pointer to an [XrSpaceContainerFB](#) structure
- `space` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

### Version History

- Revision 1, 2022-03-09 (John Schofield)
  - Initial draft
- Revision 2, 2022-05-31 (John Schofield)
  - Fix types of `XrSpaceContainerFB` fields.

## 12.71. XR\_FB\_spatial\_entity\_query

### Name String

`XR_FB_spatial_entity_query`

### Extension Type

Instance extension

### Registered Extension Number

157

### Revision

1



## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_spatial\\_entity\\_storage](#)

## Contributors

John Schofield, Facebook

Andrew Kim, Facebook

Yuichi Taguchi, Facebook

Cass Everitt, Facebook

Curtis Arink, Facebook

## Overview

This extension enables an application to discover persistent spatial entities in the area and restore them. Using the query system, the application **can** load persistent spatial entities from storage. The query system consists of a set of filters to define the spatial entity search query and an operation that needs to be performed on the search results.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SPACE\\_QUERY\\_INFO\\_FB](#)
- [XR\\_TYPE\\_SPACE\\_QUERY\\_RESULTS\\_FB](#)
- [XR\\_TYPE\\_SPACE\\_STORAGE\\_LOCATION\\_FILTER\\_INFO\\_FB](#)
- [XR\\_TYPE\\_SPACE\\_UUID\\_FILTER\\_INFO\\_FB](#)
- [XR\\_TYPE\\_SPACE\\_COMPONENT\\_FILTER\\_INFO\\_FB](#)
- [XR\\_TYPE\\_EVENT\\_DATA\\_SPACE\\_QUERY\\_RESULTS\\_AVAILABLE\\_FB](#)
- [XR\\_TYPE\\_EVENT\\_DATA\\_SPACE\\_QUERY\\_COMPLETE\\_FB](#)

## New Enums

```
// Provided by XR_FB_spatial_entity_query
typedef enum XrSpaceQueryActionFB {
    XR_SPACE_QUERY_ACTION_LOAD_FB = 0,
    XR_SPACE_QUERY_ACTION_MAX_ENUM_FB = 0x7FFFFFFF
} XrSpaceQueryActionFB;
```

Specify the type of query being performed.

## Enumerant Descriptions

- `XR_SPACE_QUERY_ACTION_LOAD_FB` — Tells the query to perform a load operation on any [XrSpace](#) returned by the query.

## New Structures

The [XrSpaceQueryInfoBaseHeaderFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceQueryInfoBaseHeaderFB {
    XrStructureType    type;
    const void*        next;
} XrSpaceQueryInfoBaseHeaderFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- `next` is `NULL` or a pointer to the next structure in a structure chain. This base structure itself has no associated [XrStructureType](#) value.

The [XrSpaceQueryInfoBaseHeaderFB](#) is a base structure that is not intended to be directly used, but forms a basis for specific query info types. All query info structures begin with the elements described in the [XrSpaceQueryInfoBaseHeaderFB](#), and a query info pointer **must** be cast to a pointer to [XrSpaceQueryInfoBaseHeaderFB](#) when passing it to the [xrQuerySpacesFB](#) function.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceQueryInfoBaseHeaderFB`
- `type` **must** be `XR_TYPE_SPACE_QUERY_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrSpaceFilterInfoBaseHeaderFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceFilterInfoBaseHeaderFB {
    XrStructureType    type;
    const void*        next;
} XrSpaceFilterInfoBaseHeaderFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. This base structure itself has no associated `XrStructureType` value.

The `XrSpaceFilterInfoBaseHeaderFB` is a base structure that is not intended to be directly used, but forms a basis for specific filter info types. All filter info structures begin with the elements described in the `XrSpaceFilterInfoBaseHeaderFB`, and a filter info pointer **must** be cast to a pointer to `XrSpaceFilterInfoBaseHeaderFB` when populating `XrSpaceQueryInfoFB::filter` and `XrSpaceQueryInfoFB::excludeFilter` to pass to the `xrQuerySpacesFB` function.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceFilterInfoBaseHeaderFB`
- `type` **must** be one of the following `XrStructureType` values: `XR_TYPE_SPACE_COMPONENT_FILTER_INFO_FB`, `XR_TYPE_SPACE_UUID_FILTER_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrSpaceStorageLocationFilterInfoFB](#)

The `XrSpaceQueryInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceQueryInfoFB {
    XrStructureType                type;
    const void*                    next;
    XrSpaceQueryActionFB           queryAction;
    uint32_t                       maxResultCount;
    XrDuration                     timeout;
    const XrSpaceFilterInfoBaseHeaderFB* filter;
    const XrSpaceFilterInfoBaseHeaderFB* excludeFilter;
} XrSpaceQueryInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `queryAction` is the type of query to perform.
- `maxResultCount` is the maximum number of entities to be found.
- `timeout` is the number of nanoseconds before the operation should time out. A value of [XR\\_INFINITE\\_DURATION](#) indicates no timeout.
- `filter` is `NULL` or a pointer to a valid structure based on [XrSpaceFilterInfoBaseHeaderFB](#).
- `excludeFilter` is `NULL` or a pointer to a valid structure based on [XrSpaceFilterInfoBaseHeaderFB](#).

May be used to query for spaces and perform a specific action on the spaces returned. The available actions are enumerated in [XrSpaceQueryActionFB](#). The filter info provided to the `filter` member of the struct is used as an inclusive filter. The filter info provided to the `excludeFilter` member of the structure is used to exclude spaces from the results returned from the filter. All spaces that match the criteria in `filter`, and that do not match the criteria in `excludeFilter`, **must** be included in the results returned. This is to allow for a more selective style query.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceQueryInfoFB`
- `type` **must** be `XR_TYPE_SPACE_QUERY_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `queryAction` **must** be a valid `XrSpaceQueryActionFB` value
- If `filter` is not `NULL`, `filter` **must** be a pointer to a valid `XrSpaceFilterInfoBaseHeaderFB`-based structure. See also: [XrSpaceComponentFilterInfoFB](#), [XrSpaceUuidFilterInfoFB](#)
- If `excludeFilter` is not `NULL`, `excludeFilter` **must** be a pointer to a valid `XrSpaceFilterInfoBaseHeaderFB`-based structure. See also: [XrSpaceComponentFilterInfoFB](#), [XrSpaceUuidFilterInfoFB](#)

The `XrSpaceStorageLocationFilterInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceStorageLocationFilterInfoFB {
    XrStructureType          type;
    const void*              next;
    XrSpaceStorageLocationFB location;
} XrSpaceStorageLocationFilterInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `location` is the location to limit the query to.

Extends a query filter to limit a query to a specific storage location. Set the `next` pointer of an `XrSpaceFilterInfoBaseHeaderFB` to chain this extra filtering functionality.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceStorageLocationFilterInfoFB`
- `type` **must** be `XR_TYPE_SPACE_STORAGE_LOCATION_FILTER_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `location` **must** be a valid `XrSpaceStorageLocationFB` value

The `XrSpaceUuidFilterInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceUuidFilterInfoFB {
    XrStructureType    type;
    const void*        next;
    uint32_t           uuidCount;
    XrUuidEXT*         uuids;
} XrSpaceUuidFilterInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `uuidCount` is the number of UUIDs to be matched.
- `uuids` is an array of `XrUuidEXT` that contains the UUIDs to be matched.

The `XrSpaceUuidFilterInfoFB` structure is a filter an application **can** use to find `XrSpace` entities that match specified UUIDs, to include or exclude them from a query.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceUuidFilterInfoFB`
- `type` **must** be `XR_TYPE_SPACE_UUID_FILTER_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `uuids` **must** be a pointer to an array of `uuidCount` `XrUuidEXT` structures
- The `uuidCount` parameter **must** be greater than `0`

The `XrSpaceComponentFilterInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceComponentFilterInfoFB {
    XrStructureType      type;
    const void*          next;
    XrSpaceComponentTypeFB componentType;
} XrSpaceComponentFilterInfoFB;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `componentType` is the `XrSpaceComponentTypeFB` to query for.

The `XrSpaceComponentFilterInfoFB` structure is a filter an application **can** use to find `XrSpace` entities which have the `componentType` enabled, to include or exclude them from a query.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceComponentFilterInfoFB`
- `type` **must** be `XR_TYPE_SPACE_COMPONENT_FILTER_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `componentType` **must** be a valid `XrSpaceComponentTypeFB` value

The `XrSpaceQueryResultFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceQueryResultFB {
    XrSpace      space;
    XrUuidEXT    uuid;
} XrSpaceQueryResultFB;
```

### Member Descriptions

- `space` is the `XrSpace` handle to the spatial entity found by the query.
- `uuid` is the UUID that identifies the entity.

The `XrSpaceQueryResultFB` structure is a query result returned in the `xrRetrieveSpaceQueryResultsFB::results` output parameter of the `xrRetrieveSpaceQueryResultsFB` function.

### Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to using `XrSpaceQueryResultFB`

The `XrSpaceQueryResultsFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrSpaceQueryResultsFB {
    XrStructureType    type;
    void*              next;
    uint32_t           resultCapacityInput;
    uint32_t           resultCountOutput;
    XrSpaceQueryResultFB* results;
} XrSpaceQueryResultsFB;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `resultCapacityInput` is the capacity of the `results` array, or 0 to indicate a request to retrieve the required capacity.
- `resultCountOutput` is an output parameter containing the count of results retrieved, or returns the required capacity in the case that `resultCapacityInput` is insufficient.
- `results` is a pointer to an array of results, but **can** be `NULL` if `resultCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `results` size.

The [XrSpaceQueryResultsFB](#) structure is used by the [xrRetrieveSpaceQueryResultsFB](#) function to retrieve query results.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_query](#) extension **must** be enabled prior to using [XrSpaceQueryResultsFB](#)
- `type` **must** be `XR_TYPE_SPACE_QUERY_RESULTS_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `resultCapacityInput` is not 0, `results` **must** be a pointer to an array of `resultCapacityInput` [XrSpaceQueryResultFB](#) structures

The [XrEventDataSpaceQueryResultsAvailableFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrEventDataSpaceQueryResultsAvailableFB {
    XrStructureType      type;
    const void*          next;
    XrAsyncRequestIdFB   requestId;
} XrEventDataSpaceQueryResultsAvailableFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous query request.

It indicates a query request has produced some number of results. If a query yields results this event **must** be delivered before the [XrEventDataSpaceQueryCompleteFB](#) event is delivered. Call [xrRetrieveSpaceQueryResultsFB](#) to retrieve those results.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_query](#) extension **must** be enabled prior to using [XrEventDataSpaceQueryResultsAvailableFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_SPACE_QUERY_RESULTS_AVAILABLE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataSpaceQueryCompleteFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_query
typedef struct XrEventDataSpaceQueryCompleteFB {
    XrStructureType    type;
    const void*        next;
    XrAsyncRequestIdFB requestId;
    XrResult            result;
} XrEventDataSpaceQueryCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous query request.
- `result` is an [XrResult](#) that determines if the request succeeded or if an error occurred.

It indicates a query request has completed and specifies the request result. This event **must** be

delivered when a query has completed, regardless of the number of results found. If any results have been found, then this event **must** be delivered after any [XrEventDataSpaceQueryResultsAvailableFB](#) events have been delivered.

### Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_query](#) extension **must** be enabled prior to using [XrEventDataSpaceQueryCompleteFB](#)
- **type** **must** be `XR_TYPE_EVENT_DATA_SPACE_QUERY_COMPLETE_FB`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### New Functions

The [xrQuerySpacesFB](#) function is defined as:

```
// Provided by XR_FB_spatial_entity_query
XrResult xrQuerySpacesFB(
    XrSession session,
    const XrSpaceQueryInfoBaseHeaderFB* info,
    XrAsyncRequestIdFB* requestId);
```

### Parameter Descriptions

- **session** is a handle to an [XrSession](#).
- **info** is a pointer to the [XrSpaceQueryInfoBaseHeaderFB](#) structure.
- **requestId** is an output parameter, and the variable it points to will be populated with the ID of this asynchronous request.

The [xrQuerySpacesFB](#) function enables an application to find and retrieve spatial entities from storage. Cast an [XrSpaceQueryInfoFB](#) pointer to a [XrSpaceQueryInfoBaseHeaderFB](#) pointer to pass as **info**. The application **should** keep the returned **requestId** for the duration of the request as it is used to refer to the request when calling [xrRetrieveSpaceQueryResultsFB](#) and is used to map completion events to the request. This operation is asynchronous and the runtime **must** post an [XrEventDataSpaceQueryCompleteFB](#) event when the operation completes successfully or encounters an error. If this function returns a failure code, no event is posted. The runtime **must** post an [XrEventDataSpaceQueryResultsAvailableFB](#) before [XrEventDataSpaceQueryCompleteFB](#) if any results are found. Once an [XrEventDataSpaceQueryResultsAvailableFB](#) event has been posted, the application **may** call [xrRetrieveSpaceQueryResultsFB](#) to retrieve the available results.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_query` extension **must** be enabled prior to calling `xrQuerySpacesFB`
- `session` **must** be a valid `XrSession` handle
- `info` **must** be a pointer to a valid `XrSpaceQueryInfoBaseHeaderFB`-based structure. See also: `XrSpaceQueryInfoFB`
- `requestId` **must** be a pointer to an `XrAsyncRequestIdFB` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrRetrieveSpaceQueryResultsFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_query
XrResult xrRetrieveSpaceQueryResultsFB(
    XrSession session,
    XrAsyncRequestIdFB requestId,
    XrSpaceQueryResultsFB* results);
```

## Parameter Descriptions

- `session` is the [XrSession](#) for which the in-progress query is valid.
- `requestId` is the [XrAsyncRequestIdFB](#) to enumerate results for.
- `results` is a pointer to an [XrSpaceQueryResultsFB](#) to populate with results.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required size of the results in this parameter.

Allows an application to retrieve all available results for a specified query. Call this function once to get the number of results found and then once more to copy the results into a buffer provided by the application. The number of results will not change between the two calls used to retrieve results. This function **must** only retrieve each query result once. After the application has used this function to retrieve a query result, the runtime frees its copy. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `requestId` refers to a request that is not yet complete, a request for which results have already been retrieved, or if `requestId` does not refer to a known request.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_query](#) extension **must** be enabled prior to calling [xrRetrieveSpaceQueryResultsFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `results` **must** be a pointer to an [XrSpaceQueryResultsFB](#) structure

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_FEATURE\_UNSUPPORTED

## Issues

## Version History

- Revision 1, 2022-01-22 (John Schofield)
  - Initial draft

## 12.72. XR\_FB\_spatial\_entity\_sharing

### Name String

XR\_FB\_spatial\_entity\_sharing

### Extension Type

Instance extension

### Registered Extension Number

170

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

and

## Contributors

John Schofield, Facebook  
Andrew Kim, Facebook

## Overview

This extension enables spatial entities to be shared between users. If the `XR_SPACE_COMPONENT_TYPE_SHARABLE_FB` component has been enabled on the spatial entity, application developers **may** share [XrSpace](#) entities between users.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SPACE_SHARE_INFO_FB`
- `XR_TYPE_EVENT_DATA_SPACE_SHARE_COMPLETE_FB`

[XrResult](#) enumeration is extended with:

- `XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB`
- `XR_ERROR_SPACE_LOCALIZATION_FAILED_FB`
- `XR_ERROR_SPACE_NETWORK_TIMEOUT_FB`
- `XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB`
- `XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB`

## New Enums

## New Base Types

## New Structures

The [XrSpaceShareInfoFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_sharing
typedef struct XrSpaceShareInfoFB {
    XrStructureType    type;
    const void*        next;
    uint32_t           spaceCount;
    XrSpace*           spaces;
    uint32_t           userCount;
    XrSpaceUserFB*     users;
} XrSpaceShareInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension. `spaceCount` is the number of elements in the `spaces` list. `spaces` is a list containing all spatial entities to be shared. `userCount` is the number of elements in the `users` list. `users` is a list of the users with which the `spaces` will: be shared.

The [XrSpaceShareInfoFB](#) structure describes a request to share one or more spatial entities with one or more users.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_sharing](#) extension **must** be enabled prior to using [XrSpaceShareInfoFB](#)
- `type` **must** be `XR_TYPE_SPACE_SHARE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `spaces` **must** be a pointer to an array of `spaceCount` [XrSpace](#) handles
- `users` **must** be a pointer to an array of `userCount` [XrSpaceUserFB](#) handles
- The `spaceCount` parameter **must** be greater than `0`
- The `userCount` parameter **must** be greater than `0`

The [XrEventDataSpaceShareCompleteFB](#) structure is defined as:



```
// Provided by XR_FB_spatial_entity_sharing
typedef struct XrEventDataSpaceShareCompleteFB {
    XrStructureType      type;
    const void*          next;
    XrAsyncRequestIdFB  requestId;
    XrResult             result;
} XrEventDataSpaceShareCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous request used to share the spatial entities.
- `result` is an [XrResult](#) that describes whether the request succeeded or if an error occurred.

It indicates that the request to share one or more spatial entities has completed. The application **can** use `result` to check if the request was successful or if an error occurred.

## Result Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB`
- `XR_ERROR_SPACE_LOCALIZATION_FAILED_FB`
- `XR_ERROR_SPACE_NETWORK_TIMEOUT_FB`
- `XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB`
- `XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB`

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_sharing` extension **must** be enabled prior to using `XrEventDataSpaceShareCompleteFB`
- `type` **must** be `XR_TYPE_EVENT_DATA_SPACE_SHARE_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The `xrShareSpacesFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_sharing
XrResult xrShareSpacesFB(
    XrSession session,
    const XrSpaceShareInfoFB* info,
    XrAsyncRequestIdFB* requestId);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `info` is a pointer to an [XrSpaceShareInfoFB](#) structure containing information about which spatial entities to share with which users.
- `requestId` is the output parameter that points to the ID of this asynchronous request.

This operation is asynchronous and the runtime **must** post an [XrEventDataSpaceShareCompleteFB](#) event when the operation completes successfully or encounters an error. If this function returns a failure code, no event is posted. The `requestId` **can** be used to later refer to the request, such as identifying which request has completed when an [XrEventDataSpaceShareCompleteFB](#) is posted to the event queue.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_sharing` extension **must** be enabled prior to calling `xrShareSpacesFB`
- `session` **must** be a valid `XrSession` handle
- `info` **must** be a pointer to a valid `XrSpaceShareInfoFB` structure
- `requestId` **must** be a pointer to an `XrAsyncRequestIdFB` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_NETWORK_TIMEOUT_FB`
- `XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB`
- `XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB`
- `XR_ERROR_SPACE_LOCALIZATION_FAILED_FB`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

## Version History

- Revision 1, 2022-06-08 (John Schofield)
  - Initial draft

# 12.73. XR\_FB\_spatial\_entity\_storage

## Name String

`XR_FB_spatial_entity_storage`

## Extension Type

Instance extension

## Registered Extension Number

159

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_FB\\_spatial\\_entity](#)

## Contributors

John Schofield, Facebook  
Andrew Kim, Facebook  
Yuichi Taguchi, Facebook  
Cass Everitt, Facebook  
Curtis Arink, Facebook

## Overview

This extension enables spatial entities to be stored and persisted across sessions. If the `XR_SPACE_COMPONENT_TYPE_STORABLE_FB` component has been enabled on the spatial entity, application developers **may** save, load, and erase persisted [XrSpace](#) entities.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SPACE_SAVE_INFO_FB`
- `XR_TYPE_SPACE_ERASE_INFO_FB`

- `XR_TYPE_EVENT_DATA_SPACE_SAVE_COMPLETE_FB`
- `XR_TYPE_EVENT_DATA_SPACE_ERASE_COMPLETE_FB`

## New Enums

```
// Provided by XR_FB_spatial_entity_storage
typedef enum XrSpaceStorageLocationFB {
    XR_SPACE_STORAGE_LOCATION_INVALID_FB = 0,
    XR_SPACE_STORAGE_LOCATION_LOCAL_FB = 1,
    XR_SPACE_STORAGE_LOCATION_CLOUD_FB = 2,
    XR_SPACE_STORAGE_LOCATION_MAX_ENUM_FB = 0x7FFFFFFF
} XrSpaceStorageLocationFB;
```

The `XrSpaceStorageLocationFB` enumeration contains the storage locations used to store, erase, and query spatial entities.

### Enumerant Descriptions

- `XR_SPACE_STORAGE_LOCATION_INVALID_FB` — Invalid storage location
- `XR_SPACE_STORAGE_LOCATION_LOCAL_FB` — Local device storage
- `XR_SPACE_STORAGE_LOCATION_CLOUD_FB` — Cloud storage

```
// Provided by XR_FB_spatial_entity_storage
typedef enum XrSpacePersistenceModeFB {
    XR_SPACE_PERSISTENCE_MODE_INVALID_FB = 0,
    XR_SPACE_PERSISTENCE_MODE_INDEFINITE_FB = 1,
    XR_SPACE_PERSISTENCE_MODE_MAX_ENUM_FB = 0x7FFFFFFF
} XrSpacePersistenceModeFB;
```

The `XrSpacePersistenceModeFB` enumeration specifies the persistence mode for the save operation.

### Enumerant Descriptions

- `XR_SPACE_PERSISTENCE_MODE_INVALID_FB` — Invalid storage persistence
- `XR_SPACE_PERSISTENCE_MODE_INDEFINITE_FB` — Store `XrSpace` indefinitely, or until erased

## New Structures

The `XrSpaceSaveInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_storage
typedef struct XrSpaceSaveInfoFB {
    XrStructureType          type;
    const void*              next;
    XrSpace                  space;
    XrSpaceStorageLocationFB location;
    XrSpacePersistenceModeFB persistenceMode;
} XrSpaceSaveInfoFB;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `space` is the `XrSpace` handle to the space of the entity to be saved.
- `location` is the storage location.
- `persistenceMode` is the persistence mode.

The `XrSpaceSaveInfoFB` structure contains information used to save the spatial entity.

### Valid Usage (Implicit)

- The `XR_FB_spatial_entity_storage` extension **must** be enabled prior to using `XrSpaceSaveInfoFB`
- `type` **must** be `XR_TYPE_SPACE_SAVE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `space` **must** be a valid `XrSpace` handle
- `location` **must** be a valid `XrSpaceStorageLocationFB` value
- `persistenceMode` **must** be a valid `XrSpacePersistenceModeFB` value

The `XrSpaceEraseInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_storage
typedef struct XrSpaceEraseInfoFB {
    XrStructureType      type;
    const void*          next;
    XrSpace               space;
    XrSpaceStorageLocationFB location;
} XrSpaceEraseInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `space` is the [XrSpace](#) handle to the reference space that defines the entity to be erased.
- `location` is the storage location.

The [XrSpaceEraseInfoFB](#) structure contains information used to erase the spatial entity.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_storage](#) extension **must** be enabled prior to using [XrSpaceEraseInfoFB](#)
- `type` **must** be `XR_TYPE_SPACE_ERASE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `space` **must** be a valid [XrSpace](#) handle
- `location` **must** be a valid [XrSpaceStorageLocationFB](#) value

The [XrEventDataSpaceSaveCompleteFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_storage
typedef struct XrEventDataSpaceSaveCompleteFB {
    XrStructureType      type;
    const void*         next;
    XrAsyncRequestIdFB  requestId;
    XrResult             result;
    XrSpace              space;
    XrUuidEXT           uuid;
    XrSpaceStorageLocationFB location;
} XrEventDataSpaceSaveCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous request to save an entity.
- `result` is an [XrResult](#) that describes whether the request succeeded or if an error occurred.
- `space` is the spatial entity being saved.
- `uuid` is the UUID for the spatial entity being saved.
- `location` is the location of the spatial entity being saved.

The save result event contains the success of the save/write operation to the specified location, as well as the [XrSpace](#) handle on which the save operation was attempted on, the unique UUID, and the triggered async request ID from the initial calling function.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_storage](#) extension **must** be enabled prior to using [XrEventDataSpaceSaveCompleteFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_SPACE_SAVE_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataSpaceEraseCompleteFB](#) structure is defined as:



```
// Provided by XR_FB_spatial_entity_storage
typedef struct XrEventDataSpaceEraseCompleteFB {
    XrStructureType      type;
    const void*         next;
    XrAsyncRequestIdFB  requestId;
    XrResult             result;
    XrSpace              space;
    XrUuidEXT           uuid;
    XrSpaceStorageLocationFB location;
} XrEventDataSpaceEraseCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous request to erase an entity.
- `result` is an [XrResult](#) that describes whether the request succeeded or if an error occurred.
- `space` is the spatial entity being erased.
- `uuid` is the UUID for the spatial entity being erased.
- `location` is the location of the spatial entity being erased.

The erase result event contains the success of the erase operation from the specified storage location. It also provides the UUID of the entity and the async request ID from the initial calling function.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_storage](#) extension **must** be enabled prior to using [XrEventDataSpaceEraseCompleteFB](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_SPACE_ERASE_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrSaveSpaceFB](#) function is defined as:

```
// Provided by XR_FB_spatial_entity_storage
XrResult xrSaveSpaceFB(
    XrSession session,
    const XrSpaceSaveInfoFB* info,
    XrAsyncRequestIdFB* requestId);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `info` contains the parameters for the save operation.
- `requestId` is an output parameter, and the variable it points to will be populated with the ID of this asynchronous request.

The `xrSaveSpaceFB` function persists the spatial entity at the specified location with the specified mode. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `XrSpaceSaveInfoFB::space` is `XR_NULL_HANDLE` or otherwise invalid. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `XrSpaceSaveInfoFB::location` or `XrSpaceSaveInfoFB::persistenceMode` is invalid. This operation is asynchronous and the runtime **must** post an `XrEventDataSpaceSaveCompleteFB` event when the operation completes successfully or encounters an error. If this function returns a failure code, no event is posted.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_storage` extension **must** be enabled prior to calling `xrSaveSpaceFB`
- `session` **must** be a valid [XrSession](#) handle
- `info` **must** be a pointer to a valid [XrSpaceSaveInfoFB](#) structure
- `requestId` **must** be a pointer to an [XrAsyncRequestIdFB](#) value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrEraseSpaceFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_storage
XrResult xrEraseSpaceFB(
    XrSession          session,
    const XrSpaceEraseInfoFB* info,
    XrAsyncRequestIdFB* requestId);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession`.
- `info` contains the parameters for the erase operation.
- `requestId` is an output parameter, and the variable it points to will be populated with the ID of this asynchronous request.

The `xrEraseSpaceFB` function erases a spatial entity from storage at the specified location. The `XrSpace` remains valid in the current session until the application destroys it or the session ends. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `XrSpaceEraseInfoFB::space` is `XR_NULL_HANDLE` or otherwise invalid. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `XrSpaceEraseInfoFB::location` is invalid. This operation is asynchronous and the runtime **must** post an

[XrEventDataSpaceEraseCompleteFB](#) event when the operation completes successfully or encounters an error. If this function returns a failure code, no event is posted.

### Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_storage](#) extension **must** be enabled prior to calling [xrEraseSpaceFB](#)
- `session` **must** be a valid [XrSession](#) handle
- `info` **must** be a pointer to a valid [XrSpaceEraseInfoFB](#) structure
- `requestId` **must** be a pointer to an [XrAsyncRequestIdFB](#) value

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

### Issues

### Version History

- Revision 1, 2022-01-22 (John Schofield)
  - Initial draft

## 12.74. XR\_FB\_spatial\_entity\_storage\_batch

### Name String

`XR_FB_spatial_entity_storage_batch`

## Extension Type

Instance extension

## Registered Extension Number

239

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_spatial\\_entity\\_storage](#)

## Contributors

John Schofield, Facebook

Andrew Kim, Facebook

## Overview

This extension enables multiple spatial entities at a time to be persisted across sessions. If the `XR_SPACE_COMPONENT_TYPE_STORABLE_FB` component has been enabled on the spatial entity, application developers **may** save and erase [XrSpace](#) entities.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SPACE_LIST_SAVE_INFO_FB`
- `XR_TYPE_EVENT_DATA_SPACE_LIST_SAVE_COMPLETE_FB`

## New Enums

## New Structures

The [XrSpaceListSaveInfoFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_storage_batch
typedef struct XrSpaceListSaveInfoFB {
    XrStructureType      type;
    const void*          next;
    uint32_t             spaceCount;
    XrSpace*             spaces;
    XrSpaceStorageLocationFB location;
} XrSpaceListSaveInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `spaceCount` is the number of spatial entities to save.
- `spaces` is a list of [XrSpace](#) handles for the entities to be saved.
- `location` is the storage location.

The [XrSpaceListSaveInfoFB](#) structure contains information used to save multiple spatial entities.

## Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity\\_storage\\_batch](#) extension **must** be enabled prior to using [XrSpaceListSaveInfoFB](#)
- `type` **must** be `XR_TYPE_SPACE_LIST_SAVE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `spaces` **must** be a pointer to an array of `spaceCount` [XrSpace](#) handles
- `location` **must** be a valid [XrSpaceStorageLocationFB](#) value
- The `spaceCount` parameter **must** be greater than `0`

The [XrEventDataSpaceListSaveCompleteFB](#) structure is defined as:

```
// Provided by XR_FB_spatial_entity_storage_batch
typedef struct XrEventDataSpaceListSaveCompleteFB {
    XrStructureType    type;
    const void*        next;
    XrAsyncRequestIdFB requestId;
    XrResult            result;
} XrEventDataSpaceListSaveCompleteFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `requestId` is the ID of the asynchronous request to save an entity.
- `result` is an [XrResult](#) that describes whether the request succeeded or if an error occurred.

This completion event indicates that a request to save a list of [XrSpace](#) objects has completed. The application **can** use `result` to check if the request was successful or if an error occurred.

## Result Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB`
- `XR_ERROR_SPACE_LOCALIZATION_FAILED_FB`
- `XR_ERROR_SPACE_NETWORK_TIMEOUT_FB`
- `XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB`
- `XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB`

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_storage_batch` extension **must** be enabled prior to using `XrEventDataSpaceListSaveCompleteFB`
- `type` **must** be `XR_TYPE_EVENT_DATA_SPACE_LIST_SAVE_COMPLETE_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The `xrSaveSpaceListFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_storage_batch
XrResult xrSaveSpaceListFB(
    XrSession session,
    const XrSpaceListSaveInfoFB* info,
    XrAsyncRequestIdFB* requestId);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `info` contains the parameters for the save operation.
- `requestId` is an output parameter, and the variable it points to will be populated with the ID of this asynchronous request.

The `xrSaveSpaceListFB` function persists the specified spatial entities at the specified storage location. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `XrSpaceSaveInfoFB::location` is invalid. This operation is asynchronous and the runtime **must** post an `XrEventDataSpaceListSaveCompleteFB` event when the operation completes successfully or encounters an error. If this function returns a failure code, no event is posted.



## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_storage_batch` extension **must** be enabled prior to calling `xrSaveSpaceListFB`
- `session` **must** be a valid `XrSession` handle
- `info` **must** be a pointer to a valid `XrSpaceListSaveInfoFB` structure
- `requestId` **must** be a pointer to an `XrAsyncRequestIdFB` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SPACE_NETWORK_TIMEOUT_FB`
- `XR_ERROR_SPACE_NETWORK_REQUEST_FAILED_FB`
- `XR_ERROR_SPACE_MAPPING_INSUFFICIENT_FB`
- `XR_ERROR_SPACE_LOCALIZATION_FAILED_FB`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_SPACE_CLOUD_STORAGE_DISABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## Issues

## Version History

- Revision 1, 2022-06-08 (John Schofield)
  - Initial draft

# 12.75. XR\_FB\_spatial\_entity\_user

## Name String

`XR_FB_spatial_entity_user`

## Extension Type

Instance extension

## Registered Extension Number

242

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

John Schofield, Facebook  
Andrew Kim, Facebook  
Andreas Selvik, Facebook

## Overview

This extension enables creation and management of user objects which **can** be used by the application to reference a user other than the current user.

In order to enable the functionality of this extension, you **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

```
XR_DEFINE_HANDLE(XrSpaceUserFB)
```

Represents a user with which the application **can** interact using various extensions including [XR\\_FB\\_spatial\\_entity\\_sharing](#). See [xrCreateSpaceUserFB](#) for how to declare a user.

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SPACE_USER_CREATE_INFO_FB`

## New Enums

## New Base Types

The `XrSpaceUserIdFB` type is defined as:

```
typedef uint64_t XrSpaceUserIdFB;
```

An implementation-defined ID of the underlying user.

## New Structures

The `XrSpaceUserCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_spatial_entity_user
typedef struct XrSpaceUserCreateInfoFB {
    XrStructureType    type;
    const void*        next;
    XrSpaceUserIdFB    userId;
} XrSpaceUserCreateInfoFB;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `userId` is the user ID with which the application can reference.

The `XrSpaceUserCreateInfoFB` structure describes a user with which the application **can** interact.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_user` extension **must** be enabled prior to using `XrSpaceUserCreateInfoFB`
- `type` **must** be `XR_TYPE_SPACE_USER_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The `xrCreateSpaceUserFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_user
XrResult xrCreateSpaceUserFB(
    XrSession session,
    const XrSpaceUserCreateInfoFB* info,
    XrSpaceUserFB* user);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `info` is a pointer to an [XrSpaceUserCreateInfoFB](#) structure containing information to create the user handle.
- `user` is the output parameter that points to the handle of the user being created.

The application **can** use this function to create a user handle with which it **can** then interact, such as sharing [XrSpace](#) objects.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_user` extension **must** be enabled prior to calling `xrCreateSpaceUserFB`
- `session` **must** be a valid [XrSession](#) handle
- `info` **must** be a pointer to a valid [XrSpaceUserCreateInfoFB](#) structure
- `user` **must** be a pointer to an [XrSpaceUserFB](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The `xrGetSpaceUserIdFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_user
XrResult xrGetSpaceUserIdFB(
    XrSpaceUserFB          user,
    XrSpaceUserIdFB*      userId);
```

## Parameter Descriptions

- `user` is a handle to an `XrSpaceUserFB`.
- `userId` is the output parameter that points to the user ID of the user.

The application **can** use this function to retrieve the user ID of a given user handle.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_user` extension **must** be enabled prior to calling `xrGetSpaceUserFB`
- `user` **must** be a valid `XrSpaceUserFB` handle
- `userId` **must** be a pointer to an `XrSpaceUserIdFB` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrDestroySpaceUserFB` function is defined as:

```
// Provided by XR_FB_spatial_entity_user
XrResult xrDestroySpaceUserFB(
    XrSpaceUserFB          user);
```

## Parameter Descriptions

- `user` is a handle to the user object to be destroyed.

The application **should** use this function to release resources tied to a given `XrSpaceUserFB` once the application no longer needs to reference the user.

## Valid Usage (Implicit)

- The `XR_FB_spatial_entity_user` extension **must** be enabled prior to calling `xrDestroySpaceUserFB`
- `user` **must** be a valid `XrSpaceUserFB` handle

## Thread Safety

- Access to `user`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`

## Issues

## Version History

- Revision 1, 2022-07-28 (John Schofield)
  - Initial draft

# 12.76. XR\_FB\_swapchain\_update\_state

## Name String

`XR_FB_swapchain_update_state`

## Extension Type

Instance extension

## Registered Extension Number

72

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

## Overview

This extension enables the application to modify and query specific mutable state associated with a swapchain.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

The [XrSwapchainStateBaseHeaderFB](#) structure is defined as:

```
// Provided by XR_FB_swapchain_update_state
typedef struct XrSwapchainStateBaseHeaderFB {
    XrStructureType    type;
    void*              next;
} XrSwapchainStateBaseHeaderFB;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure. This base structure itself has no associated [XrStructureType](#) value.
- **next** is [NULL](#) or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

The [XrSwapchainStateBaseHeaderFB](#) is a base structure that can be overridden by a specific [XrSwapchainState\\*](#) child structure.



## Valid Usage (Implicit)

- The `XR_FB_swapchain_update_state` extension **must** be enabled prior to using `XrSwapchainStateBaseHeaderFB`
- `type` **must** be one of the following `XrStructureType` values:  
`XR_TYPE_SWAPCHAIN_STATE_ANDROID_SURFACE_DIMENSIONS_FB`,  
`XR_TYPE_SWAPCHAIN_STATE_FOVEATION_FB`, `XR_TYPE_SWAPCHAIN_STATE_SAMPLER_OPENGL_ES_FB`,  
`XR_TYPE_SWAPCHAIN_STATE_SAMPLER_VULKAN_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The `xrUpdateSwapchainFB` function is defined as:

```
// Provided by XR_FB_swapchain_update_state
XrResult xrUpdateSwapchainFB(
    XrSwapchain                swapchain,
    const XrSwapchainStateBaseHeaderFB* state);
```

## Parameter Descriptions

- `swapchain` is the [XrSwapchain](#) to update state for.
- `state` is a pointer to a `XrSwapchainState` structure based off of [XrSwapchainStateBaseHeaderFB](#).

`xrUpdateSwapchainFB` provides support for an application to update specific mutable state associated with an [XrSwapchain](#).

## Valid Usage (Implicit)

- The `XR_FB_swapchain_update_state` extension **must** be enabled prior to calling `xrUpdateSwapchainFB`
- `swapchain` **must** be a valid [XrSwapchain](#) handle
- `state` **must** be a pointer to a valid [XrSwapchainStateBaseHeaderFB](#)-based structure. See also:  
[XrSwapchainStateAndroidSurfaceDimensionsFB](#), [XrSwapchainStateFoveationFB](#),  
[XrSwapchainStateSamplerOpenGLESFB](#), [XrSwapchainStateSamplerVulkanFB](#)

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrGetSwapchainStateFB` function is defined as:

```
// Provided by XR_FB_swapchain_update_state
XrResult xrGetSwapchainStateFB(
    XrSwapchain                swapchain,
    XrSwapchainStateBaseHeaderFB* state);
```

## Parameter Descriptions

- `swapchain` is the `XrSwapchain` to update state for.
- `state` is a pointer to a `XrSwapchainState` structure based off of `XrSwapchainStateBaseHeaderFB`.

`xrGetSwapchainStateFB` provides support for an application to query specific mutable state associated with an `XrSwapchain`.

## Valid Usage (Implicit)

- The `XR_FB_swapchain_update_state` extension **must** be enabled prior to calling `xrGetSwapchainStateFB`
- `swapchain` **must** be a valid `XrSwapchain` handle
- `state` **must** be a pointer to an `XrSwapchainStateBaseHeaderFB`-based structure. See also: `XrSwapchainStateAndroidSurfaceDimensionsFB`, `XrSwapchainStateFoveationFB`, `XrSwapchainStateSamplerOpenGLESFB`, `XrSwapchainStateSamplerVulkanFB`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

## Issues

- Should we add a method to query the current state?
  - Yes. Given that we allow mutable state to be updated by the application, it is useful to have a query mechanism to get the current state for all state structures.

## Version History

- Revision 1, 2021-04-16 (Gloria Kennickell)
  - Initial extension description
- Revision 2, 2021-05-13 (Gloria Kennickell)
  - Add mechanism to query current state for all state structures.
- Revision 3, 2021-05-27 (Gloria Kennickell)
  - Move platform and graphics API specific structs into separate extensions.

# 12.77. XR\_FB\_swapchain\_update\_state\_android\_surface

## Name String

`XR_FB_swapchain_update_state_android_surface`

## Extension Type

Instance extension

## Registered Extension Number

162

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_KHR\\_android\\_surface\\_swapchain](#)  
and  
[XR\\_FB\\_swapchain\\_update\\_state](#)

## Contributors

Cass Everitt, Facebook  
Gloria Kennickell, Facebook

## Overview

This extension enables the application to modify and query specific mutable state associated with an Android surface swapchain, examples include:

- A video application may need to update the default size of the image buffers associated with an Android Surface Swapchain.
- A video application may need to communicate a new width and height for an Android Surface Swapchain, as the surface dimensions may be implicitly updated by the producer during the life of the Swapchain. This is important for correct application of the non-normalized `imageRect` specified via [XrSwapchainSubImage](#).

In order to enable the functionality of this extension, the application **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo](#) `enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SWAPCHAIN\\_STATE\\_ANDROID\\_SURFACE\\_DIMENSIONS\\_FB](#)

## New Enums

## New Structures

The [XrSwapchainStateAndroidSurfaceDimensionsFB](#) structure is defined as:

```
// Provided by XR_FB_swapchain_update_state_android_surface
typedef struct XrSwapchainStateAndroidSurfaceDimensionsFB {
    XrStructureType    type;
    void*              next;
    uint32_t           width;
    uint32_t           height;
} XrSwapchainStateAndroidSurfaceDimensionsFB;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **width** is the width of the image buffer, must not be greater than the graphics API's maximum limit.
- **height** is the height of the image buffer, must not be greater than the graphics API's maximum limit.

When [XrSwapchainStateAndroidSurfaceDimensionsFB](#) is specified in the call to [xrUpdateSwapchainFB](#), the dimensions provided will be used to update the default size of the image buffers associated with the Android Surface swapchain.

Additionally, the dimensions provided will become the new source of truth for the swapchain width and height, affecting operations such as computing the normalized imageRect for the swapchain.

When [XrSwapchainStateAndroidSurfaceDimensionsFB](#) is specified in the call to [xrGetSwapchainStateFB](#), the dimensions will be populated with the current swapchain width and height.

To use [XrSwapchainStateAndroidSurfaceDimensionsFB](#), [XR\\_USE\\_PLATFORM\\_ANDROID](#) must be defined before including `openxr_platform.h`.

## Valid Usage (Implicit)

- The `XR_FB_swapchain_update_state_android_surface` extension **must** be enabled prior to using `XrSwapchainStateAndroidSurfaceDimensionsFB`
- `type` **must** be `XR_TYPE_SWAPCHAIN_STATE_ANDROID_SURFACE_DIMENSIONS_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### New Functions

### Issues

### Version History

- Revision 1, 2021-05-27 (Gloria Kennickell)
  - Initial draft

## 12.78. `XR_FB_swapchain_update_state_opengl_es`

### Name String

`XR_FB_swapchain_update_state_opengl_es`

### Extension Type

Instance extension

### Registered Extension Number

163

### Revision

1

### Extension and Version Dependencies

`XR_KHR_opengl_es_enable`

and

`XR_FB_swapchain_update_state`

### Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

### Overview

This extension enables the application to modify and query OpenGL ES-specific mutable state associated with a swapchain, examples include:

- On platforms where composition runs in a separate process from the application, swapchains must be created in a cross-process friendly way. In such cases, the texture image memory may be shared between processes, but the texture state may not; and, an explicit mechanism to synchronize this texture state between the application and the compositor is required.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo` `enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SWAPCHAIN_STATE_SAMPLER_OPENGL_ES_FB`

## New Enums

## New Structures

The `XrSwapchainStateSamplerOpenGLESTFB` structure is defined as:

```
// Provided by XR_FB_swapchain_update_state_opengl_es
typedef struct XrSwapchainStateSamplerOpenGLESTFB {
    XrStructureType    type;
    void*              next;
    EGLenum            minFilter;
    EGLenum            magFilter;
    EGLenum            wrapModeS;
    EGLenum            wrapModeT;
    EGLenum            swizzleRed;
    EGLenum            swizzleGreen;
    EGLenum            swizzleBlue;
    EGLenum            swizzleAlpha;
    float              maxAnisotropy;
    XrColor4f          borderColor;
} XrSwapchainStateSamplerOpenGLESTFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `minFilter` is a valid Android OpenGL ES [GLenum](#).
- `magFilter` is a valid Android OpenGL ES [GLenum](#).
- `wrapModeS` is a valid Android OpenGL ES [GLenum](#).
- `wrapModeT` is a valid Android OpenGL ES [GLenum](#).
- `swizzleRed` is a valid Android OpenGL ES [GLenum](#).
- `swizzleGreen` is a valid Android OpenGL ES [GLenum](#).
- `swizzleBlue` is a valid Android OpenGL ES [GLenum](#).
- `swizzleAlpha` is a valid Android OpenGL ES [GLenum](#).
- `maxAnisotropy` is a valid float used to represent max anisotropy.
- `borderColor` is an RGBA color to be used as border texels.

When [XrSwapchainStateSamplerOpenGLESFB](#) is specified in the call to [xrUpdateSwapchainFB](#), texture sampler state for all images in the [XrSwapchain](#) will be updated for both the application and compositor processes.

For most cases, the sampler state update is only required compositor-side, as that is where the swapchain images are sampled. For completeness, the application-side sampler state is additionally updated to support cases where the application may choose to directly sample the swapchain images.

Applications are expected to handle synchronization of the sampler state update with application-side rendering. Similarly, the compositor will synchronize the sampler state update with rendering of the next compositor frame.

An [EGLContext](#), either the [EGLContext](#) bound during [XrSwapchain](#) creation or an [EGLContext](#) in the same share group, is required to be bound on the application calling thread. Current texture bindings may be altered by the call, including the active texture.

When [XrSwapchainStateSamplerOpenGLESFB](#) is specified in the call to [xrGetSwapchainStateFB](#), the sampler state will be populated with the current swapchain sampler state.

To use [XrSwapchainStateSamplerOpenGLESFB](#), [XR\\_USE\\_GRAPHICS\\_API\\_OPENGL\\_ES](#) must be defined before including [openxr\\_platform.h](#).



## Valid Usage (Implicit)

- The `XR_FB_swapchain_update_state_opengl_es` extension **must** be enabled prior to using `XrSwapchainStateSamplerOpenGLESFB`
- `type` **must** be `XR_TYPE_SWAPCHAIN_STATE_SAMPLER_OPENGL_ES_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `minFilter` **must** be a valid `EGLenum` value
- `magFilter` **must** be a valid `EGLenum` value
- `wrapModeS` **must** be a valid `EGLenum` value
- `wrapModeT` **must** be a valid `EGLenum` value
- `swizzleRed` **must** be a valid `EGLenum` value
- `swizzleGreen` **must** be a valid `EGLenum` value
- `swizzleBlue` **must** be a valid `EGLenum` value
- `swizzleAlpha` **must** be a valid `EGLenum` value

## New Functions

## Issues

## Version History

- Revision 1, 2021-05-27 (Gloria Kennickell)
  - Initial draft

# 12.79. XR\_FB\_swapchain\_update\_state\_vulkan

## Name String

`XR_FB_swapchain_update_state_vulkan`

## Extension Type

Instance extension

## Registered Extension Number

164

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_KHR\\_vulkan\\_enable](#)

and

[XR\\_FB\\_swapchain\\_update\\_state](#)

## Contributors

Cass Everitt, Facebook

Gloria Kennickell, Facebook

## Overview

This extension enables the application to modify and query Vulkan-specific mutable state associated with a swapchain, examples include:

- On platforms where composition runs in a separate process from the application, swapchains must be created in a cross-process friendly way. In such cases, the texture image memory may be shared between processes, but the texture state may not; and, an explicit mechanism to synchronize this texture state between the application and the compositor is required.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo](#) `enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SWAPCHAIN_STATE_SAMPLER_VULKAN_FB`

## New Enums

## New Structures

The [XrSwapchainStateSamplerVulkanFB](#) structure is defined as:

```
// Provided by XR_FB_swapchain_update_state_vulkan
typedef struct XrSwapchainStateSamplerVulkanFB {
    XrStructureType      type;
    void*                next;
    VkFilter              minFilter;
    VkFilter              magFilter;
    VkSamplerMipmapMode  mipmapMode;
    VkSamplerAddressMode wrapModeS;
    VkSamplerAddressMode wrapModeT;
    VkComponentSwizzle   swizzleRed;
    VkComponentSwizzle   swizzleGreen;
    VkComponentSwizzle   swizzleBlue;
    VkComponentSwizzle   swizzleAlpha;
    float                maxAnisotropy;
    XrColor4f            borderColor;
} XrSwapchainStateSamplerVulkanFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `minFilter` is a valid Vulkan [VkFilter](#).
- `magFilter` is a valid Vulkan [VkFilter](#).
- `mipmapMode` is a valid Vulkan [VkSamplerMipmapMode](#).
- `wrapModeS` is a valid Vulkan [VkSamplerAddressMode](#).
- `wrapModeT` is a valid Vulkan [VkSamplerAddressMode](#).
- `swizzleRed` is a valid Vulkan [VkComponentSwizzle](#).
- `swizzleGreen` is a valid Vulkan [VkComponentSwizzle](#).
- `swizzleBlue` is a valid Vulkan [VkComponentSwizzle](#).
- `swizzleAlpha` is a valid Vulkan [VkComponentSwizzle](#).
- `maxAnisotropy` is a valid float used to represent max anisotropy.
- `borderColor` is an RGBA color to be used as border texels.

When [XrSwapchainStateSamplerVulkanFB](#) is specified in the call to [xrUpdateSwapchainFB](#), texture sampler state for all images in the [XrSwapchain](#) will be updated for the compositor process. For most cases, the sampler state update is only required compositor-side, as that is where the swapchain images are sampled. If the application requires sampling of the swapchain images, the application will be responsible for updating the texture state using normal Vulkan mechanisms and synchronizing

appropriately with application-side rendering.

When `XrSwapchainStateSamplerVulkanFB` is specified in the call to `xrGetSwapchainStateFB`, the sampler state will be populated with the current swapchain sampler state.

To use `XrSwapchainStateSamplerVulkanFB`, `XR_USE_GRAPHICS_API_VULKAN` must be defined before including `openxr_platform.h`.

### Valid Usage (Implicit)

- The `XR_FB_swapchain_update_state_vulkan` extension **must** be enabled prior to using `XrSwapchainStateSamplerVulkanFB`
- `type` **must** be `XR_TYPE_SWAPCHAIN_STATE_SAMPLER_VULKAN_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `minFilter` **must** be a valid `VkFilter` value
- `magFilter` **must** be a valid `VkFilter` value
- `mipmapMode` **must** be a valid `VkSamplerMipmapMode` value
- `wrapModeS` **must** be a valid `VkSamplerAddressMode` value
- `wrapModeT` **must** be a valid `VkSamplerAddressMode` value
- `swizzleRed` **must** be a valid `VkComponentSwizzle` value
- `swizzleGreen` **must** be a valid `VkComponentSwizzle` value
- `swizzleBlue` **must** be a valid `VkComponentSwizzle` value
- `swizzleAlpha` **must** be a valid `VkComponentSwizzle` value

### New Functions

### Issues

### Version History

- Revision 1, 2021-05-27 (Gloria Kennickell)
  - Initial draft

## 12.80. XR\_FB\_touch\_controller\_pro

### Name String

`XR_FB_touch_controller_pro`

### Extension Type

Instance extension

## Registered Extension Number

168

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-06-29

## IP Status

No known IP claims.

## Contributors

Aanchal Dalmia, Meta

Adam Bengis, Meta

Tony Targonski, Meta

Federico Schliemann, Meta

## Overview

This extension defines a new interaction profile for the Meta Quest Touch Pro Controller.

Meta Quest Touch Pro Controller Profile Path:

- */interaction\_profiles/facebook/touch\_controller\_pro*

### Note

The interaction profile path */interaction\_profiles/facebook/touch\_controller\_pro* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/facebook/touch\_controller\_pro\_fb*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile provides inputs and outputs that are a superset of those available in the existing "Oculus Touch Controller" interaction profile:

- */interaction\_profiles/oculus/touch\_controller*

Supported component paths (Note that the paths which are marked as 'new' are enabled by Meta Quest Touch Pro Controller profile exclusively):

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
  - *.../input/system/click* (**may** not be available for application use)
- On both:
  - *.../input/squeeze/value*
  - *.../input/trigger/value*
  - *.../input/trigger/touch*
  - *.../input/thumbstick*
  - *.../input/thumbstick/x*
  - *.../input/thumbstick/y*
  - *.../input/thumbstick/click*
  - *.../input/thumbstick/touch*
  - *.../input/thumbrest/touch*
  - *.../input/grip/pose*
  - *.../input/aim/pose*
  - *.../output/haptic*
  - *.../input/thumbrest/force* (new)
  - *.../input/stylus\_fb/force* (new)
  - *.../input/trigger/curl\_fb* (new)
  - *.../input/trigger/slide\_fb* (new)
  - *.../input/trigger/proximity\_fb* (new)

- `.../input/thumb_fb/proximity_fb` (new)
- `.../output/haptic_trigger_fb` (new)
- `.../output/haptic_thumb_fb` (new)

## New Identifiers

- **stylus\_fb**: Meta Quest Touch Pro Controller adds an optional stylus tip that can be interchanged with the lanyard. This tip can detect various pressure levels and could be used for writing or drawing.
- **thumb\_fb**: Meta Quest Touch Pro Controller adds a 1-dimensional analog input value for the thumb. This is similar to other triggers on the controller like the fore trigger for the index finger and grip trigger for the middle finger.

## Input Path Descriptions

- `/input/thumbrest/force` : Allow developers to access the normalized 1D force value associated with the thumb ranging from 0-6 Newtons: 0 = not pressed, 1 = fully pressed
- `/input/stylus_fb/force` : Allow developers to access the normalized 1D force value associated with the stylus ranging from ~0-2 Newtons: 0 = not pressed, 1 = fully pressed
- `/input/trigger/curl_fb` : This represents how pointed or curled the user's finger is on the trigger: 0 = fully pointed, 1 = finger flat on surface
- `/input/trigger/slide_fb`: This represents how far the user is sliding their index finger along the surface of the trigger: 0 = finger flat on the surface, 1 = finger fully drawn back
- `/input/trigger/proximity_fb` : Bit indicating whether the user's index finger is near the trigger
- `/input/thumb_fb/proximity_fb` : Bit indicating the user's thumb is near the touchpad

## Output Path Descriptions

In addition to the VCM motor, Meta Quest Touch Pro Controller has two localized LRA haptics elements located in the fore trigger and under the touchpad.

- `/output/haptic_trigger_fb` represents the path to the haptic element in the trigger
- `/output/haptic_thumb_fb` represents the path to the haptic element under the touchpad

## Version History

- Revision 1, 2022-06-29 (Aanchal Dalmia)

- Initial extension proposal

## 12.81. XR\_FB\_touch\_controller\_proximity

### Name String

`XR_FB_touch_controller_proximity`

### Extension Type

Instance extension

### Registered Extension Number

207

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-09-12

### IP Status

No known IP claims.

### Contributors

Tony Targonski, Meta Platforms  
Aanchal Dalmia, Meta Platforms  
Andreas Loeve Selvik, Meta Platforms  
John Kearney, Meta Platforms  
James Hillery, Meta Platforms

### 12.81.1. Overview

This extension introduces a new component path, `proximity_fb`, and adds support for it for the `/interaction_profiles/oculus/touch_controller` interaction profile.

### 12.81.2. New Interaction Profile Component Paths

- `proximity_fb` - The user is in physical proximity of input source. This **may** be present for any kind of input source representing a physical component, such as a button, if the device includes the necessary sensor. The state of a "proximity\_fb" component **must** be `XR_TRUE` if the same input source is returning `XR_TRUE` for either a "touch" or any other component that implies physical contact. The runtime **may** return `XR_TRUE` for "proximity\_fb" when "touch" returns `XR_FALSE`. This indicate that the user is hovering just above, but not touching the input source in question.



"proximity\_fb" components are always boolean.

### 12.81.3. Interaction Profile Changes

Interaction profile: */interaction\_profiles/oculus/touch\_controller*

Additional supported component paths for the above profile enabled by this extension:

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

On both:

- *.../input/trigger/proximity\_fb* This represents whether the user is in proximity of the trigger button, usually with their index finger.
- *.../input/thumb\_fb/proximity\_fb* This represents whether the user is in proximity of the input sources at the top of the controller, usually with their thumb.

### 12.81.4. Example code

The following example code demonstrates detecting when a user lifts their finger off the trigger button.

```
XrInstance instance;           // previously initialized
XrSession session;           // previously initialized
XrActionSet inGameActionSet;  // previously initialized

XrAction indexProximityAction; // previously initialized
XrAction indexTouchAction;    // previously initialized

// -----
// Bind actions to trigger/proximity_fb and trigger/touch
// -----

XrPath indexProximityPath, indexTouchPath;
// New component exposed by this extension:
CHK_XR(xrStringToPath(instance, "/user/hand/right/input/trigger/proximity_fb",
&indexProximityPath));
// Existing component that is useful together with proximity_fb
CHK_XR(xrStringToPath(instance, "/user/hand/right/input/trigger/touch", &indexTouchPath))

XrPath interactionProfilePath;
CHK_XR(xrStringToPath(instance, "/interaction_profiles/oculus/touch_controller",
&interactionProfilePath));
```

```

XrActionSuggestedBinding bindings[2];
bindings[0].action = indexProximityAction;
bindings[0].binding = indexProximityPath;
bindings[1].action = indexTouchAction;
bindings[1].binding = indexTouchPath;

XrInteractionProfileSuggestedBinding
suggestedBindings{XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING};
suggestedBindings.interactionProfile = interactionProfilePath;
suggestedBindings.suggestedBindings = bindings;
suggestedBindings.countSuggestedBindings = 2;
CHK_XR(xrSuggestInteractionProfileBindings(instance, &suggestedBindings));

// -----
// Application main loop
// -----

while (1)
{
    // ...

    // -----
    // Query input state
    // -----

    XrActionStateBoolean indexTouchState{XR_TYPE_ACTION_STATE_BOOLEAN};
    XrActionStateBoolean indexProximityState{XR_TYPE_ACTION_STATE_BOOLEAN};
    XrActionStateGetInfo getInfo{XR_TYPE_ACTION_STATE_GET_INFO};

    getInfo.action = indexTouchAction;
    CHK_XR(xrGetActionStateBoolean(session, &getInfo, &indexTouchState));
    getInfo.action = indexProximityAction;
    CHK_XR(xrGetActionStateBoolean(session, &getInfo, &indexProximityState));

    // -----
    // Proximity and touch logic
    // -----

    // There are only three valid combinations of the proximity and touch values
    if (!indexProximityState.currentState)
    {
        // Index is not in proximity of the trigger button (they might be pointing!)
        // Implies that TouchState.currentState == XR_FALSE
    }
    if (indexProximityState.currentState && !indexTouchState.currentState)
    {
        // Index finger of user is in proximity of, but not touching, the trigger button

```

```
    // i.e. they are hovering above the button
  }
  if (indexTouchState.currentState)
  {
    // Index finger of user is touching the trigger button
    // Implies that ProximityState.currentState == XR_TRUE
  }
}
```

### **New Object Types**

### **New Flag Types**

### **New Enum Constants**

### **New Enums**

### **New Structures**

### **New Object Types**

### **New Flag Types**

### **New Enum Constants**

### **New Enums**

### **New Structures**

### **Version History**

- Revision 1, 2022-09-12 (Andreas Loeve Selvik)
  - Initial extension proposal

## **12.82. XR\_FB\_triangle\_mesh**

### **Name String**

`XR_FB_triangle_mesh`

### **Extension Type**

Instance extension

### **Registered Extension Number**

118

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Anton Vaneev, Facebook

Cass Everitt, Facebook

Federico Schliemann, Facebook

Johannes Schmid, Facebook

## Overview

Meshes may be useful in XR applications when representing parts of the environment. In particular, application may provide the surfaces of real-world objects tagged manually to the runtime, or obtain automatically detected environment contents.

This extension allows:

- An application to create a triangle mesh and specify the mesh data.
- An application to update mesh contents if a mesh is mutable.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## New Object Types

```
XR_DEFINE_HANDLE(XrTriangleMeshFB)
```

`XrTriangleMeshFB` represents a triangle mesh with its corresponding mesh data: a vertex buffer and an index buffer.

## New Flag Types

```
// Provided by XR_FB_triangle_mesh  
typedef XrFlags64 XrTriangleMeshFlagsFB;
```

```
// Flag bits for XrTriangleMeshFlagsFB
static const XrTriangleMeshFlagsFB XR_TRIANGLE_MESH_MUTABLE_BIT_FB = 0x00000001;
```

## Flag Descriptions

- `XR_TRIANGLE_MESH_MUTABLE_BIT_FB` — The triangle mesh is mutable (can be modified after it is created).

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_TRIANGLE_MESH_CREATE_INFO_FB`

## New Enums

Applications may specify the triangle winding order of a mesh - whether the vertices of an outward-facing side of a triangle appear in clockwise or counter-clockwise order - using `XrWindingOrderFB` enumeration.

```
// Provided by XR_FB_triangle_mesh
typedef enum XrWindingOrderFB {
    XR_WINDING_ORDER_UNKNOWN_FB = 0,
    XR_WINDING_ORDER_CW_FB = 1,
    XR_WINDING_ORDER_CCW_FB = 2,
    XR_WINDING_ORDER_MAX_ENUM_FB = 0x7FFFFFFF
} XrWindingOrderFB;
```

## Enumerant Descriptions

- `XR_WINDING_ORDER_UNKNOWN_FB` — Winding order is unknown and the runtime cannot make any assumptions on the triangle orientation
- `XR_WINDING_ORDER_CW_FB` — Clockwise winding order
- `XR_WINDING_ORDER_CCW_FB` — Counter-clockwise winding order

## New Structures

`XrTriangleMeshCreateInfoFB` **must** be provided when calling `xrCreateTriangleMeshFB`.

The `XrTriangleMeshCreateInfoFB` structure is defined as:

```
// Provided by XR_FB_triangle_mesh
typedef struct XrTriangleMeshCreateInfoFB {
    XrStructureType      type;
    const void*         next;
    XrTriangleMeshFlagsFB flags;
    XrWindingOrderFB    windingOrder;
    uint32_t            vertexCount;
    const XrVector3f*   vertexBuffer;
    uint32_t            triangleCount;
    const uint32_t*     indexBuffer;
} XrTriangleMeshCreateInfoFB;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of [XrTriangleMeshFlagBitsFB](#) that specify behavior.
- `windingOrder` is the [XrWindingOrderFB](#) value defining the winding order of the mesh triangles.
- `vertexCount` is the number of vertices in the mesh. In the case of the mutable mesh, the value is treated as the maximum number of vertices the mesh will be able to represent at any time in its lifecycle. The actual number of vertices can vary and is defined when [xrTriangleMeshEndUpdateFB](#) is called.
- `vertexBuffer` is a pointer to the vertex data. The size of the array must be `vertexCount` elements. When the mesh is mutable (`(flags & XR_TRIANGLE_MESH_MUTABLE_BIT_FB) != 0`), the `vertexBuffer` parameter **must** be `NULL` and mesh data **must** be populated separately.
- `triangleCount` is the number of triangles in the mesh. In the case of the mutable mesh, the value is treated as the maximum number of triangles the mesh will be able to represent at any time in its lifecycle. The actual number of triangles can vary and is defined when [xrTriangleMeshEndUpdateFB](#) is called.
- `indexBuffer` the triangle indices. The size of the array must be `triangleCount` elements. When the mesh is mutable (`(flags & XR_TRIANGLE_MESH_MUTABLE_BIT_FB) != 0`), the `indexBuffer` parameter **must** be `NULL` and mesh data **must** be populated separately.

Mesh buffers **can** be updated between [xrTriangleMeshBeginUpdateFB](#) and [xrTriangleMeshEndUpdateFB](#) calls.

If the mesh is non-mutable, `vertexBuffer` **must** be a pointer to an array of `vertexCount` [XrVector3f](#)

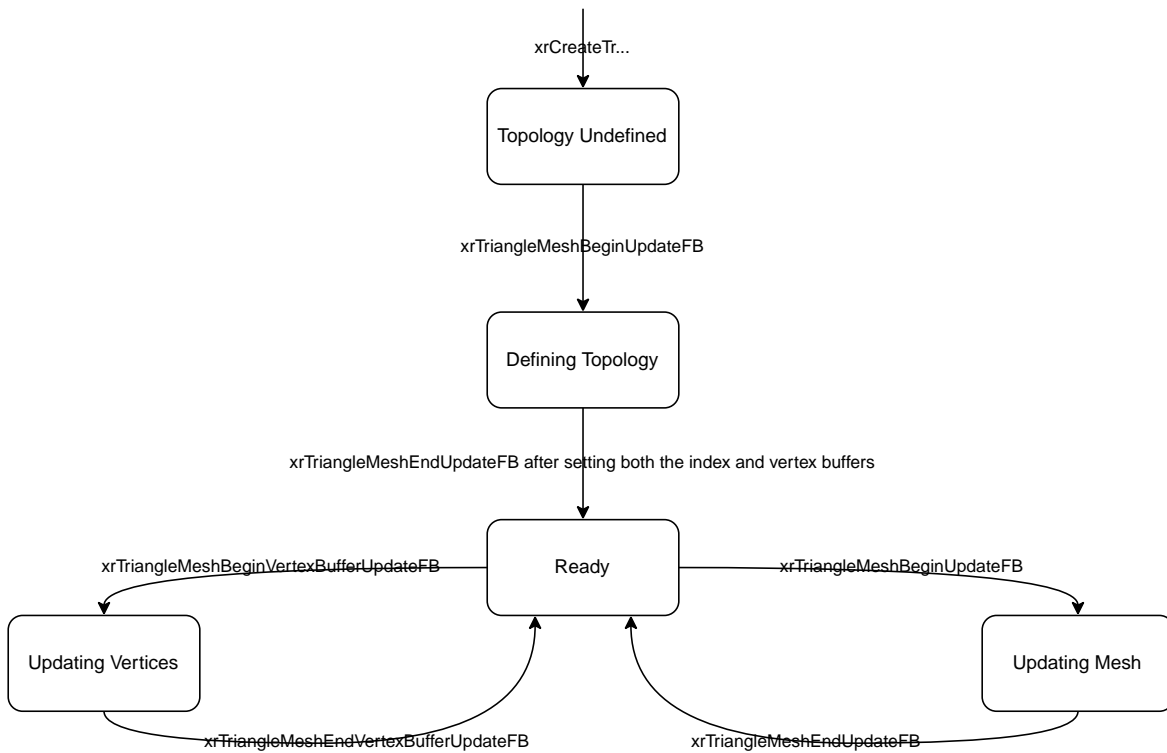
structures. If the mesh is non-mutable, `indexBuffer` **must** be a pointer to an array of  $3 * \text{triangleCount}$  `uint32_t` vertex indices.

### Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to using `XrTriangleMeshCreateInfoFB`
- `type` **must** be `XR_TYPE_TRIANGLE_MESH_CREATE_INFO_FB`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be `0` or a valid combination of `XrTriangleMeshFlagBitsFB` values
- `windingOrder` **must** be a valid `XrWindingOrderFB` value

### Mutable Mesh Update States

Mutable meshes have a state machine controlling how they may be updated.



May change vertex buffer contents, but no...

May change vertex buffer size and/or cont...

Viewer does not support full SVG 1.1

Figure 15. Mutable Triangle Mesh States

The states are as follows:

## Undefined Topology

The default state immediately after creation of a mutable mesh. Move to [Defining Topology](#) by calling `xrTriangleMeshBeginUpdateFB`.

## Defining Topology

The application **must** set the initial vertex buffer and index buffer before moving to [Ready](#) by calling `xrTriangleMeshEndUpdateFB`.

## Ready

In this state, the buffer contents/size **must** not be modified. To move to [Updating Mesh](#) call `xrTriangleMeshBeginUpdateFB`. To move to [Updating Vertices](#) call `xrTriangleMeshBeginVertexBufferUpdateFB`.

## Updating Mesh

The application **may** modify the vertex buffer contents and/or the vertex count. The application **may** modify the index buffer contents and/or the index buffer element count. Move to [Ready](#) and commit changes by calling `xrTriangleMeshEndUpdateFB`.

## Updating Vertices

The application **may** modify the vertex buffer contents, but not the vertex count. Move to [Ready](#) and commit changes by calling `xrTriangleMeshEndVertexBufferUpdateFB`.

## New Functions

The `xrCreateTriangleMeshFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrCreateTriangleMeshFB(
    XrSession session,
    const XrTriangleMeshCreateInfoFB* createInfo,
    XrTriangleMeshFB* outTriangleMesh);
```

### Parameter Descriptions

- `session` is the [XrSession](#) to which the mesh will belong.
- `createInfo` is a pointer to an [XrTriangleMeshCreateInfoFB](#) structure containing parameters to be used to create the mesh.
- `outTriangleMesh` is a pointer to a handle in which the created [XrTriangleMeshFB](#) is returned.

This creates an [XrTriangleMeshFB](#) handle. The returned triangle mesh handle **may** be subsequently used in API calls.



When the mesh is mutable (the `XR_TRIANGLE_MESH_MUTABLE_BIT_FB` bit is set in `XrTriangleMeshCreateInfoFB::flags`), the created triangle mesh starts in the `Undefined Topology` state.

Immutable meshes have no state machine; they may be considered to be in state `Ready` with no valid edges leaving that state.

### Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to calling `xrCreateTriangleMeshFB`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrTriangleMeshCreateInfoFB` structure
- `outTriangleMesh` **must** be a pointer to an `XrTriangleMeshFB` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_INSUFFICIENT_RESOURCES_PASSTHROUGH_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrDestroyTriangleMeshFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrDestroyTriangleMeshFB(
    XrTriangleMeshFB          mesh);
```

## Parameter Descriptions

- `mesh` is the [XrTriangleMeshFB](#) to destroy.

[XrTriangleMeshFB](#) handles and their associated data are destroyed by [xrDestroyTriangleMeshFB](#). The mesh buffers retrieved by [xrTriangleMeshGetVertexBufferFB](#) and [xrTriangleMeshGetIndexBufferFB](#) **must** not be accessed anymore after their parent mesh object has been destroyed.

## Valid Usage (Implicit)

- The [XR\\_FB\\_triangle\\_mesh](#) extension **must** be enabled prior to calling [xrDestroyTriangleMeshFB](#)
- `mesh` **must** be a valid [XrTriangleMeshFB](#) handle

## Thread Safety

- Access to `mesh`, and any child handles, **must** be externally synchronized
- Access to the buffers returned from calls to [xrTriangleMeshGetVertexBufferFB](#) and [xrTriangleMeshGetIndexBufferFB](#) on `mesh` **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The [xrTriangleMeshGetVertexBufferFB](#) function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrTriangleMeshGetVertexBufferFB(
    XrTriangleMeshFB          mesh,
    XrVector3f**              outVertexBuffer);
```

## Parameter Descriptions

- `mesh` is the [XrTriangleMeshFB](#) to get the vertex buffer for.
- `outVertexBuffer` is a pointer to return the vertex buffer into.

Retrieves a pointer to the vertex buffer. The vertex buffer is structured as an array of [XrVector3f](#). The size of the buffer is [XrTriangleMeshCreateInfoFB::vertexCount](#) elements. The buffer location is guaranteed to remain constant over the lifecycle of the mesh object.

A mesh **must** be mutable and in a specific state for the application to **modify** it through the retrieved vertex buffer.

- A mutable triangle mesh **must** be in state [Defining Topology](#), [Updating Mesh](#), or [Updating Vertices](#) to modify the **contents** of the vertex buffer retrieved by this function.
- A mutable triangle mesh **must** be in state [Defining Topology](#) or [Updating Mesh](#) to modify the **count** of elements in the vertex buffer retrieved by this function. The new count is passed as a parameter to [xrTriangleMeshEndUpdateFB](#).

## Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to calling [xrTriangleMeshGetVertexBufferFB](#)
- `mesh` **must** be a valid [XrTriangleMeshFB](#) handle
- `outVertexBuffer` **must** be a pointer to a pointer to an [XrVector3f](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrTriangleMeshGetIndexBufferFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrTriangleMeshGetIndexBufferFB(
    XrTriangleMeshFB          mesh,
    uint32_t**                outIndexBuffer);
```

## Parameter Descriptions

- `mesh` is the `XrTriangleMeshFB` to get the index buffer for.
- `outIndexBuffer` is a pointer to return the index buffer into.

Retrieves a pointer to the index buffer that defines the topology of the triangle mesh. Each triplet of consecutive elements points to three vertices in the vertex buffer and thus form a triangle. The size of the index buffer is  $3 * \text{XrTriangleMeshCreateInfoFB::triangleCount}$  elements. The buffer location is guaranteed to remain constant over the lifecycle of the mesh object.

A triangle mesh **must** be mutable and in state `Defining Topology` or `Updating Mesh` for the application to **modify** the contents and/or triangle count in the index buffer retrieved by this function.

## Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to calling `xrTriangleMeshGetIndexBufferFB`
- `mesh` **must** be a valid `XrTriangleMeshFB` handle
- `outIndexBuffer` **must** be a pointer to a pointer to a `uint32_t` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrTriangleMeshBeginUpdateFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrTriangleMeshBeginUpdateFB(
    XrTriangleMeshFB          mesh);
```

## Parameter Descriptions

- `mesh` is the `XrTriangleMeshFB` to update.

Begins updating the mesh buffer data. The application **must** call this function before it makes any modifications to the buffers retrieved by `xrTriangleMeshGetVertexBufferFB` and `xrTriangleMeshGetIndexBufferFB`. If only the vertex buffer contents need to be updated, and the mesh

is in state [Ready](#), `xrTriangleMeshBeginVertexBufferUpdateFB` **may** be used instead. To commit the modifications, the application **must** call `xrTriangleMeshEndUpdateFB`.

The triangle mesh `mesh` **must** be mutable. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the mesh is immutable.

The triangle mesh `mesh` **must** be in state [Undefined Topology](#) or [Ready](#).

- If the triangle mesh is in state [Undefined Topology](#) before this call, a successful call moves it to state [Defining Topology](#).
- If the triangle mesh is in state [Ready](#) before this call, a successful call moves it to state [Updating Mesh](#).

### Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to calling `xrTriangleMeshBeginUpdateFB`
- `mesh` **must** be a valid `XrTriangleMeshFB` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_CALL_ORDER_INVALID`

The `xrTriangleMeshEndUpdateFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrTriangleMeshEndUpdateFB(
    XrTriangleMeshFB          mesh,
    uint32_t                  vertexCount,
    uint32_t                  triangleCount);
```

## Parameter Descriptions

- `mesh` is the [XrTriangleMeshFB](#) to update.
- `vertexCount` is the vertex count after the update.
- `triangleCount` is the triangle count after the update.

Signals to the runtime that the application has finished initially populating or updating the mesh buffers. `vertexCount` and `triangleCount` specify the actual number of primitives that make up the mesh after the update. They **must** be larger than zero but smaller or equal to the maximum counts defined at create time. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if an invalid count is passed.

The triangle mesh `mesh` **must** be mutable. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the mesh is immutable.

The triangle mesh `mesh` **must** be in state [Defining Topology](#) or [Updating Mesh](#).

A successful call moves `mesh` to state [Ready](#).

## Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to calling [xrTriangleMeshEndUpdateFB](#)
- `mesh` **must** be a valid [XrTriangleMeshFB](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_CALL_ORDER_INVALID`

The `xrTriangleMeshBeginVertexBufferUpdateFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrTriangleMeshBeginVertexBufferUpdateFB(
    XrTriangleMeshFB          mesh,
    uint32_t*                 outVertexCount);
```

## Parameter Descriptions

- `mesh` is the `XrTriangleMeshFB` to update.
- `outVertexCount` is a pointer to a value to populate with the current vertex count. The updated data must have the exact same number of vertices.

Begins an update of the vertex positions of a mutable triangle mesh. The vertex count returned through `outVertexCount` is defined by the last call to `xrTriangleMeshEndUpdateFB`. Once the modification is done, call `xrTriangleMeshEndVertexBufferUpdateFB` to commit the changes and move to state `Ready`.

The triangle mesh `mesh` **must** be mutable. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the mesh is immutable.



The triangle mesh `mesh` **must** be in state `Ready`.

A successful call moves `mesh` to state `Updating Vertices`.

### Valid Usage (Implicit)

- The `XR_FB_triangle_mesh` extension **must** be enabled prior to calling `xrTriangleMeshBeginVertexBufferUpdateFB`
- `mesh` **must** be a valid `XrTriangleMeshFB` handle
- `outVertexCount` **must** be a pointer to a `uint32_t` value

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_CALL_ORDER_INVALID`

The `xrTriangleMeshEndVertexBufferUpdateFB` function is defined as:

```
// Provided by XR_FB_triangle_mesh
XrResult xrTriangleMeshEndVertexBufferUpdateFB(
    XrTriangleMeshFB mesh);
```

## Parameter Descriptions

- `mesh` is the [XrTriangleMeshFB](#) to update.

Signals to the runtime that the application has finished updating the vertex buffer data following a call to [xrTriangleMeshBeginVertexBufferUpdateFB](#).

The triangle mesh `mesh` **must** be mutable. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the mesh is immutable.

The triangle mesh `mesh` **must** be in state [Updating Vertices](#).

A successful call moves `mesh` to state [Ready](#).

## Valid Usage (Implicit)

- The [XR\\_FB\\_triangle\\_mesh](#) extension **must** be enabled prior to calling [xrTriangleMeshEndVertexBufferUpdateFB](#)
- `mesh` **must** be a valid [XrTriangleMeshFB](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`
- `XR_ERROR_CALL_ORDER_INVALID`

## Issues

## Version History

- Revision 1, 2021-09-01 (Anton Vaneev)
  - Initial extension description
- Revision 2, 2022-01-07 (Rylie Pavlik, Collabora, Ltd.)
  - Add a state diagram to clarify valid usage, and allow `XR_ERROR_CALL_ORDER_INVALID`.

## 12.83. XR\_HTC\_anchor

### Name String

`XR_HTC_anchor`

### Extension Type

Instance extension

### Registered Extension Number

320

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-09-14

### IP Status

No known IP claims.

### Contributors

CheHsuan Shu, HTC

Bill Chang, HTC

### Overview

This extension allows an application to create a spatial anchor to track a point in the physical environment. The runtime adjusts the pose of the anchor over time to align it with the real world.

### Inspect system capability

The `XrSystemAnchorPropertiesHTC` structure is defined as:

```
// Provided by XR_HTC_anchor
typedef struct XrSystemAnchorPropertiesHTC {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsAnchor;
} XrSystemAnchorPropertiesHTC;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsAnchor` indicates if current system is capable of anchor functionality.

An application **can** inspect whether the system is capable of anchor functionality by chaining an `XrSystemAnchorPropertiesHTC` structure to the `XrSystemProperties` when calling `xrGetSystemProperties`. The runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if `XrSystemAnchorPropertiesHTC::supportsAnchor` was `XR_FALSE`.

## Valid Usage (Implicit)

- The `XR_HTC_anchor` extension **must** be enabled prior to using `XrSystemAnchorPropertiesHTC`
- `type` **must** be `XR_TYPE_SYSTEM_ANCHOR_PROPERTIES_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `xrCreateSpatialAnchorHTC` function is defined as:

```
// Provided by XR_HTC_anchor
XrResult xrCreateSpatialAnchorHTC(
    XrSession                session,
    const XrSpatialAnchorCreateInfoHTC* createInfo,
    XrSpace*                 anchor);
```

## Parameter Descriptions

- `session` is the [XrSession](#) to create the anchor in.
- `createInfo` is the [XrSpatialAnchorCreateInfoHTC](#) used to specify the anchor.
- `anchor` is the returned [XrSpace](#) handle.

The [xrCreateSpatialAnchorHTC](#) function creates a spatial anchor with specified base space and pose in the space. The anchor is represented by an [XrSpace](#) and its pose can be tracked via [xrLocateSpace](#). Once the anchor is no longer needed, call [xrDestroySpace](#) to erase the anchor.

## Valid Usage (Implicit)

- The [XR-HTC-anchor](#) extension **must** be enabled prior to calling [xrCreateSpatialAnchorHTC](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrSpatialAnchorCreateInfoHTC](#) structure
- `anchor` **must** be a pointer to an [XrSpace](#) handle

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_OUT\\_OF\\_MEMORY](#)
- [XR\\_ERROR\\_LIMIT\\_REACHED](#)
- [XR\\_ERROR\\_POSE\\_INVALID](#)
- [XR\\_ERROR\\_NAME\\_INVALID](#)

The [XrSpatialAnchorCreateInfoHTC](#) structure is defined as:

```
// Provided by XR_HTC_anchor
typedef struct XrSpatialAnchorCreateInfoHTC {
    XrStructureType      type;
    const void*          next;
    XrSpace               space;
    XrPosef              poseInSpace;
    XrSpatialAnchorNameHTC name;
} XrSpatialAnchorCreateInfoHTC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `space` is the [XrSpace](#) in which `poseInSpace` is specified.
- `poseInSpace` is the [XrPosef](#) specifying the point in the real world within `space`.
- `name` is the [XrSpatialAnchorNameHTC](#) containing the name of the anchor.

The `poseInSpace` is transformed into world space to specify the point in the real world. The anchor tracks changes of the reality and **may** not be affected by the changes of `space`.

## Valid Usage (Implicit)

- The [XR\\_HTC\\_anchor](#) extension **must** be enabled prior to using [XrSpatialAnchorCreateInfoHTC](#)
- `type` **must** be `XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `space` **must** be a valid [XrSpace](#) handle
- `name` **must** be a valid [XrSpatialAnchorNameHTC](#) structure

The [XrSpatialAnchorNameHTC](#) structure is defined as:

```
// Provided by XR_HTC_anchor
typedef struct XrSpatialAnchorNameHTC {
    char    name[XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_HTC];
} XrSpatialAnchorNameHTC;
```

## Member Descriptions

- `name` is a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_HTC`.

## Valid Usage (Implicit)

- The `XR_HTC_anchor` extension **must** be enabled prior to using `XrSpatialAnchorNameHTC`
- `name` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_HTC`

The `xrGetSpatialAnchorNameHTC` function is defined as:

```
// Provided by XR_HTC_anchor
XrResult xrGetSpatialAnchorNameHTC(
    XrSpace                                anchor,
    XrSpatialAnchorNameHTC*                name);
```

## Parameter Descriptions

- `anchor` is the `XrSpace` created by `xrCreateSpatialAnchorHTC`.
- `name` is a pointer to output `XrSpatialAnchorNameHTC`.

The `xrGetSpatialAnchorNameHTC` function gets the name of an anchor. If the provided `anchor` is a valid space handle but was **not** created with `xrCreateSpatialAnchorHTC`, the runtime **must** return `XR_ERROR_NOT_AN_ANCHOR_HTC`.

## Valid Usage (Implicit)

- The `XR_HTC_anchor` extension **must** be enabled prior to calling `xrGetSpatialAnchorNameHTC`
- `anchor` **must** be a valid `XrSpace` handle
- `name` **must** be a pointer to an `XrSpatialAnchorNameHTC` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_NOT_AN_ANCHOR_HTC`

## New Object Types

## New Flag Types

## New Enum Constants

- `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_HTC`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SYSTEM_ANCHOR_PROPERTIES_HTC`
- `XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_HTC`

`XrResult` enumeration is extended with:

- `XR_ERROR_NOT_AN_ANCHOR_HTC`

## New Enums



## New Structures

- [XrSystemAnchorPropertiesHTC](#)
- [XrSpatialAnchorCreateInfoHTC](#)
- [XrSpatialAnchorNameHTC](#)

## New Functions

- [xrCreateSpatialAnchorHTC](#)
- [xrGetSpatialAnchorNameHTC](#)

## Issues

## Version History

- Revision 1, 2023-09-14 (CheHsuan Shu)
  - Initial extension description

# 12.84. XR\_HTC\_facial\_tracking

## Name String

`XR_HTC_facial_tracking`

## Extension Type

Instance extension

## Registered Extension Number

105

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-12-16

## IP Status

No known IP claims.

## Contributors

Kyle Chen, HTC  
Chris Kuo

## Overview

This extension allows an application to track and integrate users' eye and lip movements, empowering developers to read intention and model facial expressions.

## Inspect system capability

[XrSystemFacialTrackingPropertiesHTC](#) is defined as:

```
// Provided by XR_HTC_facial_tracking
typedef struct XrSystemFacialTrackingPropertiesHTC {
    XrStructureType    type;
    void*              next;
    XrBool32           supportEyeFacialTracking;
    XrBool32           supportLipFacialTracking;
} XrSystemFacialTrackingPropertiesHTC;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportEyeFacialTracking` indicates if the current system is capable of generating eye expressions.
- `supportLipFacialTracking` indicates if the current system is capable of generating lip expressions.

An application **can** inspect whether the system is capable of two of the facial tracking by extending the [XrSystemProperties](#) with [XrSystemFacialTrackingPropertiesHTC](#) structure when calling [xrGetSystemProperties](#).

### Valid Usage (Implicit)

- The [XR\\_HTC\\_facial\\_tracking](#) extension **must** be enabled prior to using [XrSystemFacialTrackingPropertiesHTC](#)
- `type` **must** be `XR_TYPE_SYSTEM_FACIAL_TRACKING_PROPERTIES_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

If a runtime returns `XR_FALSE` for `supportEyeFacialTracking`, the runtime **must** return

`XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateFacialTrackerHTC` with `XR_FACIAL_TRACKING_TYPE_EYE_DEFAULT_HTC` set for `XrFacialTrackingTypeHTC` in `XrFacialTrackerCreateInfoHTC`. Similarly, if a runtime returns `XR_FALSE` for `supportLipFacialTracking` the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateFacialTrackerHTC` with `XR_FACIAL_TRACKING_TYPE_LIP_DEFAULT_HTC` set for `XrFacialTrackingTypeHTC` in `XrFacialTrackerCreateInfoHTC`.

### Create a facial tracker handle

The `XrFacialTrackerHTC` handle represents the resources for an facial tracker of the specific facial tracking type.

```
XR_DEFINE_HANDLE(XrFacialTrackerHTC)
```

An application creates separate `XrFacialTrackerHTC` handles for eye tracker or lip tracker. This handle can be used to retrieve corresponding facial expressions using `xrGetFacialExpressionsHTC` function.

The `xrCreateFacialTrackerHTC` function is defined as

```
// Provided by XR_HTC_facial_tracking
XrResult xrCreateFacialTrackerHTC(
    XrSession session,
    const XrFacialTrackerCreateInfoHTC* createInfo,
    XrFacialTrackerHTC* facialTracker);
```

### Parameter Descriptions

- `session` is an `XrSession` in which the facial expression will be active.
- `createInfo` is the `XrFacialTrackerCreateInfoHTC` used to specify the facial tracking type.
- `facialTracker` is the returned `XrFacialTrackerHTC` handle.

An application **can** create an `XrFacialTrackerHTC` handle using `xrCreateFacialTrackerHTC`.

If the system does not support eye tracking or lip tracking, runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateFacialTrackerHTC` according to the corresponding case. In this case, the runtime **must** return `XR_FALSE` for `XrSystemFacialTrackingPropertiesHTC::supportEyeFacialTracking` or `XrSystemFacialTrackingPropertiesHTC::supportLipFacialTracking` when the function `xrGetSystemProperties` is called, so that the application **may** avoid creating a facial tracker.

## Valid Usage (Implicit)

- The `XR_HTC_facial_tracking` extension **must** be enabled prior to calling `xrCreateFacialTrackerHTC`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrFacialTrackerCreateInfoHTC` structure
- `facialTracker` **must** be a pointer to an `XrFacialTrackerHTC` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrFacialTrackerCreateInfoHTC` structure is defined as:

```
// Provided by XR_HTC_facial_tracking
typedef struct XrFacialTrackerCreateInfoHTC {
    XrStructureType          type;
    const void*              next;
    XrFacialTrackingTypeHTC facialTrackingType;
} XrFacialTrackerCreateInfoHTC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `facialTrackingType` is an [XrFacialTrackingTypeHTC](#) which describes which type of facial tracking should be used for this handle.

The [XrFacialTrackerCreateInfoHTC](#) structure describes the information to create an [XrFacialTrackerHTC](#) handle.

## Valid Usage (Implicit)

- The [XR\\_HTC\\_facial\\_tracking](#) extension **must** be enabled prior to using [XrFacialTrackerCreateInfoHTC](#)
- `type` **must** be `XR_TYPE_FACIAL_TRACKER_CREATE_INFO_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `facialTrackingType` **must** be a valid [XrFacialTrackingTypeHTC](#) value

The [XrFacialTrackingTypeHTC](#) describes which type of tracking the [XrFacialTrackerHTC](#) is using.

```
// Provided by XR_HTC_facial_tracking
typedef enum XrFacialTrackingTypeHTC {
    XR_FACIAL_TRACKING_TYPE_EYE_DEFAULT_HTC = 1,
    XR_FACIAL_TRACKING_TYPE_LIP_DEFAULT_HTC = 2,
    XR_FACIAL_TRACKING_TYPE_MAX_ENUM_HTC = 0x7FFFFFFF
} XrFacialTrackingTypeHTC;
```

## Enumerant Descriptions

- `XR_FACIAL_TRACKING_TYPE_EYE_DEFAULT_HTC` — Specifies this handle will observe eye expressions, with values indexed by [XrEyeExpressionHTC](#) whose count is [XR\\_FACIAL\\_EXPRESSION\\_EYE\\_COUNT\\_HTC](#).
- `XR_FACIAL_TRACKING_TYPE_LIP_DEFAULT_HTC` — Specifies this handle will observe lip expressions, with values indexed by [XrLipExpressionHTC](#) whose count is [XR\\_FACIAL\\_EXPRESSION\\_LIP\\_COUNT\\_HTC](#).

The `xrDestroyFacialTrackerHTC` function is defined as:

```
// Provided by XR_HTC_facial_tracking
XrResult xrDestroyFacialTrackerHTC(
    XrFacialTrackerHTC          facialTracker);
```

### Parameter Descriptions

- `facialTracker` is an `XrFacialTrackerHTC` previously created by `xrCreateFacialTrackerHTC`.

`xrDestroyFacialTrackerHTC` releases the `facialTracker` and the underlying resources when finished with facial tracking experiences.

### Valid Usage (Implicit)

- The `XR_HTC_facial_tracking` extension **must** be enabled prior to calling `xrDestroyFacialTrackerHTC`
- `facialTracker` **must** be a valid `XrFacialTrackerHTC` handle

### Thread Safety

- Access to `facialTracker`, and any child handles, **must** be externally synchronized

### Return Codes

#### Success

- `XR_SUCCESS`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## Retrieve facial expressions

The `xrGetFacialExpressionsHTC` function is defined as:

```
// Provided by XR_HTC_facial_tracking
XrResult xrGetFacialExpressionsHTC(
    XrFacialTrackerHTC          facialTracker,
    XrFacialExpressionsHTC*    facialExpressions);
```

## Parameter Descriptions

- `facialTracker` is an `XrFacialTrackerHTC` previously created by `xrCreateFacialTrackerHTC`.
- `facialExpressions` is a pointer to `XrFacialExpressionsHTC` receiving the returned facial expressions.

`xrGetFacialExpressionsHTC` retrieves an array of values of blend shapes for a facial expression on a given time.

## Valid Usage (Implicit)

- The `XR_HTC_facial_tracking` extension **must** be enabled prior to calling `xrGetFacialExpressionsHTC`
- `facialTracker` **must** be a valid `XrFacialTrackerHTC` handle
- `facialExpressions` **must** be a pointer to an `XrFacialExpressionsHTC` structure

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_TIME_INVALID`

The `XrFacialExpressionsHTC` structure is defined as:

```
// Provided by XR_HTC_facial_tracking
typedef struct XrFacialExpressionsHTC {
    XrStructureType    type;
    const void*        next;
    XrBool32           isActive;
    XrTime              sampleTime;
    uint32_t           expressionCount;
    float*             expressionWeightings;
} XrFacialExpressionsHTC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `isActive` is an [XrBool32](#) indicating if the facial tracker is active.
- `sampleTime` is when in time the expression is expressed.
- `expressionCount` is a [uint32\\_t](#) describing the count of elements in `expressionWeightings` array.
- `expressionWeightings` is a `float` array filled in by the runtime, specifying the weightings for each blend shape.

[XrFacialExpressionsHTC](#) structure returns data of a lip facial expression or an eye facial expression.

An application **must** preallocate the output `expressionWeightings` array that can contain at least `expressionCount` of `float`. `expressionCount` **must** be at least [XR\\_FACIAL\\_EXPRESSION\\_LIP\\_COUNT\\_HTC](#) for [XR\\_FACIAL\\_TRACKING\\_TYPE\\_LIP\\_DEFAULT\\_HTC](#), and at least [XR\\_FACIAL\\_EXPRESSION\\_EYE\\_COUNT\\_HTC](#) for [XR\\_FACIAL\\_TRACKING\\_TYPE\\_EYE\\_DEFAULT\\_HTC](#).

The application **must** set `expressionCount` as described by the [XrFacialTrackingTypeHTC](#) when creating the [XrFacialTrackerHTC](#) otherwise the runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#).

The runtime **must** update the `expressionWeightings` array ordered so that the application can index elements using the corresponding facial tracker enum (e.g. [XrEyeExpressionHTC](#) or [XrLipExpressionHTC](#)) as described by [XrFacialTrackingTypeHTC](#) when creating the [XrFacialTrackerHTC](#). For example, when the [XrFacialTrackerHTC](#) is created with [XrFacialTrackerHTC::facialTrackingType](#) set to [XR\\_FACIAL\\_TRACKING\\_TYPE\\_EYE\\_DEFAULT\\_HTC](#), the application **must** set the `expressionCount` to [XR\\_FACIAL\\_EXPRESSION\\_EYE\\_COUNT\\_HTC](#), and the runtime **must** fill the `expressionWeightings` array ordered with eye expression data so that it can be indexed by the [XrEyeExpressionHTC](#) enum.

If the returned `isActive` is true, the runtime **must** fill the `expressionWeightings` array ordered.



If the returned `isActive` is false, it indicates the facial tracker did not detect the corresponding facial input or the application lost input focus.

If the input `expressionCount` is not sufficient to contain all output indices, the runtime **must** return `XR_ERROR_SIZE_INSUFFICIENT` on calls to `xrGetFacialExpressionsHTC` and not change the content in `expressionWeightings`.

### Valid Usage (Implicit)

- The `XR_HTC_facial_tracking` extension **must** be enabled prior to using `XrFacialExpressionsHTC`
- `type` **must** be `XR_TYPE_FACIAL_EXPRESSIONS_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `expressionWeightings` **must** be a pointer to a `float` value

```
// Provided by XR_HTC_facial_tracking
#define XR_FACIAL_EXPRESSION_EYE_COUNT_HTC 14
```

The number of blend shapes in an expression of type `XR_FACIAL_TRACKING_TYPE_EYE_DEFAULT_HTC`.

```
// Provided by XR_HTC_facial_tracking
#define XR_FACIAL_EXPRESSION_LIP_COUNT_HTC 37
```

The number of blend shapes in an expression of type `XR_FACIAL_TRACKING_TYPE_LIP_DEFAULT_HTC`.

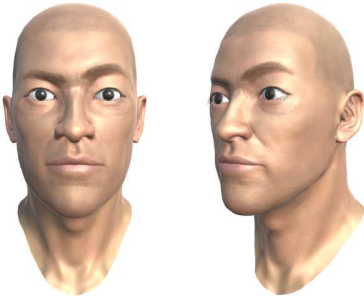
### Facial Expression List

- **Eye Blend Shapes**

Through feeding the blend shape values of eye expression to an avatar, its facial expression can be animated with the player's eye movement. The following pictures show how the facial expression acts on the avatar according to each set of eye blend shape values.

```
// Provided by XR_HTC_facial_tracking
typedef enum XrEyeExpressionHTC {
    XR_EYE_EXPRESSION_LEFT_BLINK_HTC = 0,
    XR_EYE_EXPRESSION_LEFT_WIDE_HTC = 1,
    XR_EYE_EXPRESSION_RIGHT_BLINK_HTC = 2,
    XR_EYE_EXPRESSION_RIGHT_WIDE_HTC = 3,
    XR_EYE_EXPRESSION_LEFT_SQUEEZE_HTC = 4,
    XR_EYE_EXPRESSION_RIGHT_SQUEEZE_HTC = 5,
    XR_EYE_EXPRESSION_LEFT_DOWN_HTC = 6,
    XR_EYE_EXPRESSION_RIGHT_DOWN_HTC = 7,
    XR_EYE_EXPRESSION_LEFT_OUT_HTC = 8,
    XR_EYE_EXPRESSION_RIGHT_IN_HTC = 9,
    XR_EYE_EXPRESSION_LEFT_IN_HTC = 10,
    XR_EYE_EXPRESSION_RIGHT_OUT_HTC = 11,
    XR_EYE_EXPRESSION_LEFT_UP_HTC = 12,
    XR_EYE_EXPRESSION_RIGHT_UP_HTC = 13,
    XR_EYE_EXPRESSION_MAX_ENUM_HTC = 0x7FFFFFFF
} XrEyeExpressionHTC;
```

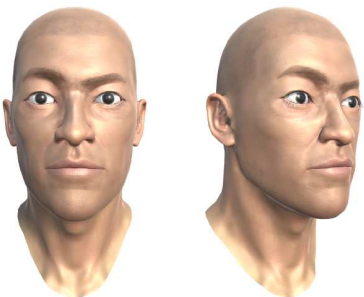
#### XR\_EYE\_EXPRESSION\_LEFT\_WIDE\_HTC



##### Description

This blend shape keeps left eye wide and at that time `XR_EYE_EXPRESSION_LEFT_BLINK_HTC` value is 0.

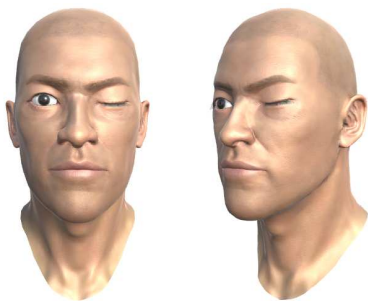
#### XR\_EYE\_EXPRESSION\_RIGHT\_WIDE\_HTC



##### Description

This blend shape keeps right eye wide and at that time `XR_EYE_EXPRESSION_RIGHT_BLINK_HTC` value is 0.

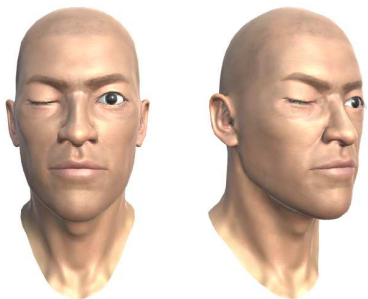
### XR\_EYE\_EXPRESSION\_LEFT\_BLINK\_HTC



#### Description

This blend shape influences blinking of the right eye. When this value goes higher, left eye approaches close.

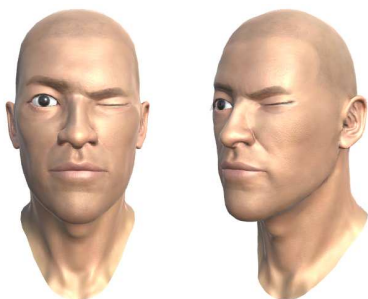
### XR\_EYE\_EXPRESSION\_RIGHT\_BLINK\_HTC



#### Description

This blend shape influences blinking of the right eye. When this value goes higher, right eye approaches close.

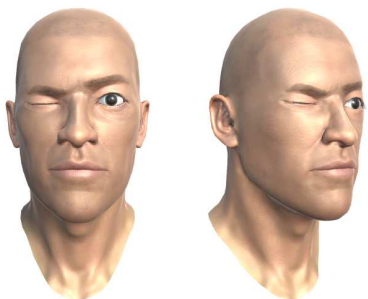
### XR\_EYE\_EXPRESSION\_LEFT\_SQUEEZE\_HTC



#### Description

The blend shape closes eye tightly and at that time  
`XR_EYE_EXPRESSION_LEFT_BLINK_HTC` value is 1.

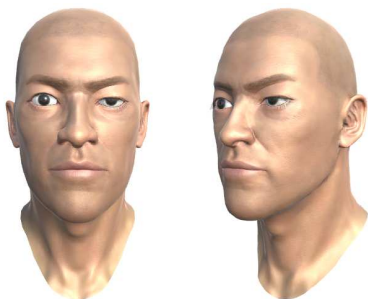
### XR\_EYE\_EXPRESSION\_RIGHT\_SQUEEZE\_HTC



#### Description

The blend shape closes eye tightly and at that time  
`XR_EYE_EXPRESSION_RIGHT_BLINK_HTC` value is 1.

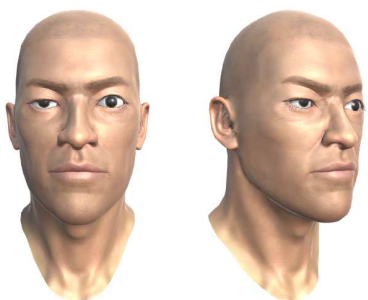
#### XR\_EYE\_EXPRESSION\_LEFT\_DOWN\_HTC



##### **Description**

This blendShape influences the muscles around the left eye, moving these muscles further downward with a higher value.

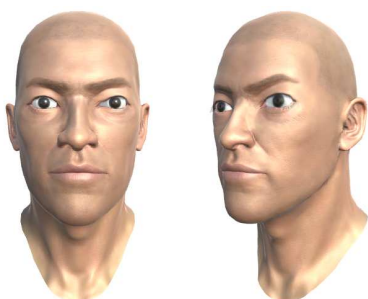
#### XR\_EYE\_EXPRESSION\_RIGHT\_DOWN\_HTC



##### **Description**

This blendShape influences the muscles around the right eye, moving these muscles further downward with a higher value.

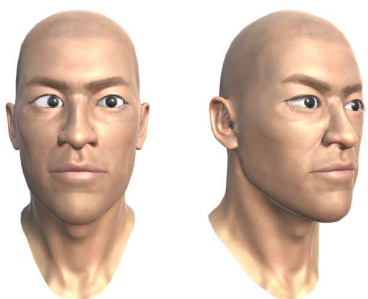
#### XR\_EYE\_EXPRESSION\_LEFT\_OUT\_HTC



##### **Description**

This blendShape influences the muscles around the left eye, moving these muscles further leftward with a higher value.

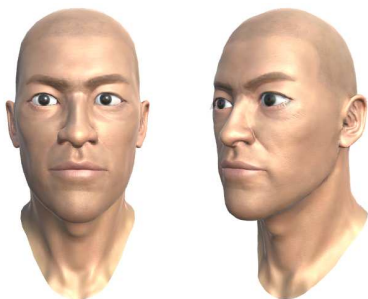
#### XR\_EYE\_EXPRESSION\_RIGHT\_IN\_HTC



##### **Description**

This blendShape influences the muscles around the right eye, moving these muscles further leftward with a higher value.

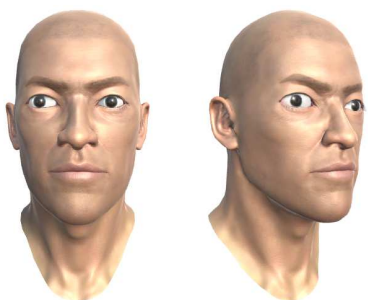
#### XR\_EYE\_EXPRESSION\_LEFT\_IN\_HTC



##### **Description**

This blendShape influences the muscles around the left eye, moving these muscles further rightward with a higher value.

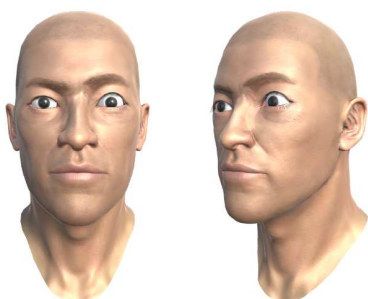
#### XR\_EYE\_EXPRESSION\_RIGHT\_OUT\_HTC



##### **Description**

This blendShape influences the muscles around the right eye, moving these muscles further rightward with a higher value.

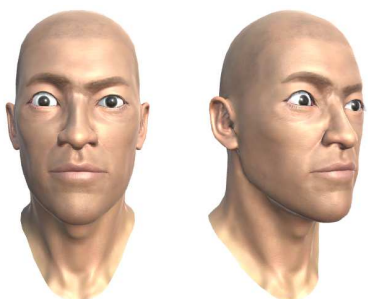
#### XR\_EYE\_EXPRESSION\_LEFT\_UP\_HTC



##### **Description**

This blendShape influences the muscles around the left eye, moving these muscles further upward with a higher value.

#### XR\_EYE\_EXPRESSION\_RIGHT\_UP\_HTC



##### **Description**

This blendShape influences the muscles around the right eye, moving these muscles further upward with a higher value.

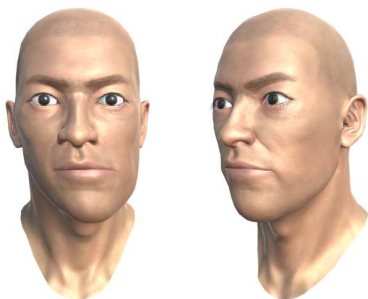
#### • Lip Blend Shapes

Through feeding the blend shape values of lip expression to an avatar, its facial expression can be

animated with the player's lip movement. The following pictures show how the facial expression acts on the avatar according to each set of lip blend shape values.

```
// Provided by XR_HTC_facial_tracking
typedef enum XrLipExpressionHTC {
    XR_LIP_EXPRESSION_JAW_RIGHT_HTC = 0,
    XR_LIP_EXPRESSION_JAW_LEFT_HTC = 1,
    XR_LIP_EXPRESSION_JAW_FORWARD_HTC = 2,
    XR_LIP_EXPRESSION_JAW_OPEN_HTC = 3,
    XR_LIP_EXPRESSION_MOUTH_APE_SHAPE_HTC = 4,
    XR_LIP_EXPRESSION_MOUTH_UPPER_RIGHT_HTC = 5,
    XR_LIP_EXPRESSION_MOUTH_UPPER_LEFT_HTC = 6,
    XR_LIP_EXPRESSION_MOUTH_LOWER_RIGHT_HTC = 7,
    XR_LIP_EXPRESSION_MOUTH_LOWER_LEFT_HTC = 8,
    XR_LIP_EXPRESSION_MOUTH_UPPER_OVERTURN_HTC = 9,
    XR_LIP_EXPRESSION_MOUTH_LOWER_OVERTURN_HTC = 10,
    XR_LIP_EXPRESSION_MOUTH_POUT_HTC = 11,
    XR_LIP_EXPRESSION_MOUTH_SMILE_RIGHT_HTC = 12,
    XR_LIP_EXPRESSION_MOUTH_SMILE_LEFT_HTC = 13,
    XR_LIP_EXPRESSION_MOUTH_SAD_RIGHT_HTC = 14,
    XR_LIP_EXPRESSION_MOUTH_SAD_LEFT_HTC = 15,
    XR_LIP_EXPRESSION_CHEEK_PUFF_RIGHT_HTC = 16,
    XR_LIP_EXPRESSION_CHEEK_PUFF_LEFT_HTC = 17,
    XR_LIP_EXPRESSION_CHEEK_SUCK_HTC = 18,
    XR_LIP_EXPRESSION_MOUTH_UPPER_UPRIGHT_HTC = 19,
    XR_LIP_EXPRESSION_MOUTH_UPPER_UPLEFT_HTC = 20,
    XR_LIP_EXPRESSION_MOUTH_LOWER_DOWNRIGHT_HTC = 21,
    XR_LIP_EXPRESSION_MOUTH_LOWER_DOWNLEFT_HTC = 22,
    XR_LIP_EXPRESSION_MOUTH_UPPER_INSIDE_HTC = 23,
    XR_LIP_EXPRESSION_MOUTH_LOWER_INSIDE_HTC = 24,
    XR_LIP_EXPRESSION_MOUTH_LOWER_OVERLAY_HTC = 25,
    XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC = 26,
    XR_LIP_EXPRESSION_TONGUE_LEFT_HTC = 27,
    XR_LIP_EXPRESSION_TONGUE_RIGHT_HTC = 28,
    XR_LIP_EXPRESSION_TONGUE_UP_HTC = 29,
    XR_LIP_EXPRESSION_TONGUE_DOWN_HTC = 30,
    XR_LIP_EXPRESSION_TONGUE_ROLL_HTC = 31,
    XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC = 32,
    XR_LIP_EXPRESSION_TONGUE_UPRIGHT_MORPH_HTC = 33,
    XR_LIP_EXPRESSION_TONGUE_UPLEFT_MORPH_HTC = 34,
    XR_LIP_EXPRESSION_TONGUE_DOWNRIGHT_MORPH_HTC = 35,
    XR_LIP_EXPRESSION_TONGUE_DOWNLEFT_MORPH_HTC = 36,
    XR_LIP_EXPRESSION_MAX_ENUM_HTC = 0x7FFFFFFF
} XrLipExpressionHTC;
```

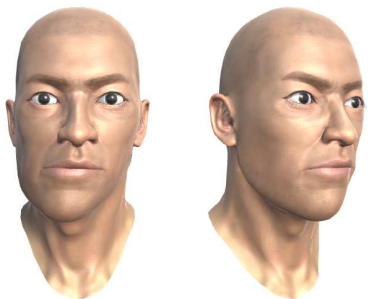
XR\_LIP\_EXPRESSION\_JAW\_LEFT\_HTC



**Description**

This blend shape moves the jaw further leftward with a higher value.

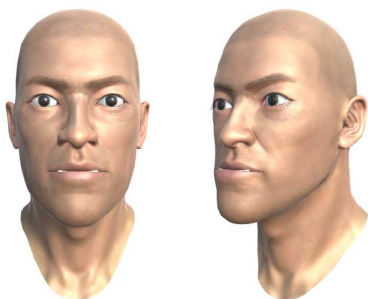
XR\_LIP\_EXPRESSION\_JAW\_RIGHT\_HTC



**Description**

This blend shape moves the jaw further rightward with a higher value.

XR\_LIP\_EXPRESSION\_JAW\_FORWARD\_HTC



**Description**

This blend shape moves the jaw forward with a higher value.

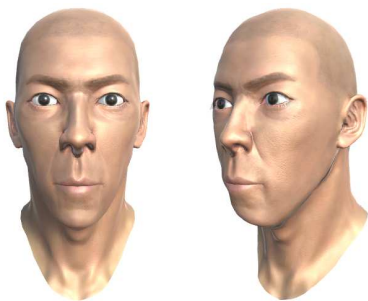
XR\_LIP\_EXPRESSION\_JAW\_OPEN\_HTC



**Description**

This blend shape opens the mouth further with a higher value.

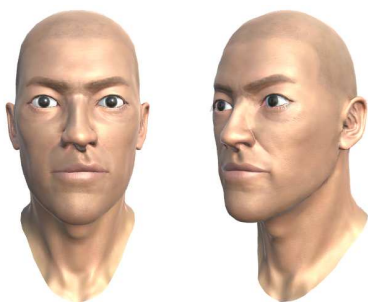
XR\_LIP\_EXPRESSION\_MOUTH\_APE\_SHAPE\_HTC



**Description**

This blend shape stretches the jaw further with a higher value.

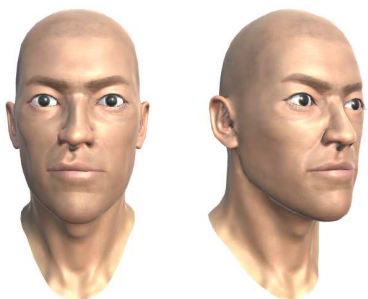
XR\_LIP\_EXPRESSION\_MOUTH\_UPPER\_LEFT\_HTC



**Description**

This blend shape moves your upper lip leftward.

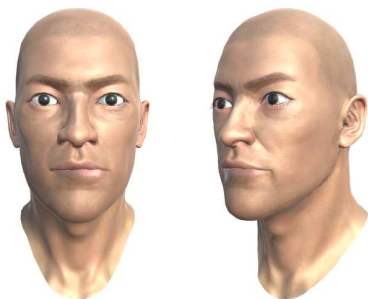
XR\_LIP\_EXPRESSION\_MOUTH\_UPPER\_RIGHT\_HTC



**Description**

This blend shape moves your upper lip rightward.

XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_LEFT\_HTC

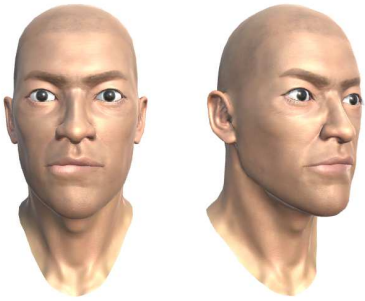


**Description**

This blend shape moves your lower lip leftward.



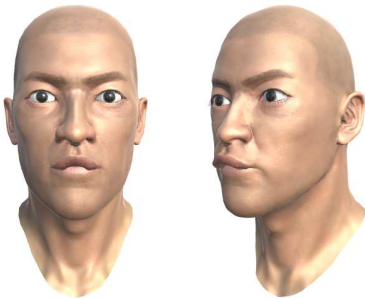
#### XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_RIGHT\_HTC



##### **Description**

This blend shape moves your lower lip rightward.

#### XR\_LIP\_EXPRESSION\_MOUTH\_UPPER\_OVERTURN\_HTC



##### **Description**

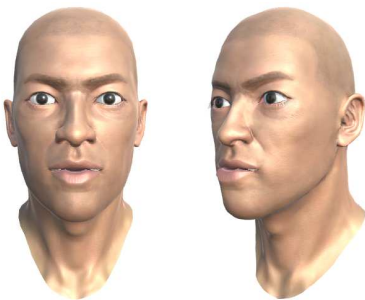
This blend shape pouts your upper lip.

Can be used with

[XR\\_LIP\\_EXPRESSION\\_MOUTH\\_UPPER\\_UPRIGHT\\_HTC](#) and

[XR\\_LIP\\_EXPRESSION\\_MOUTH\\_UPPER\\_UPLEFT\\_HTC](#) to complete upper O mouth shape.

#### XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_OVERTURN\_HTC



##### **Description**

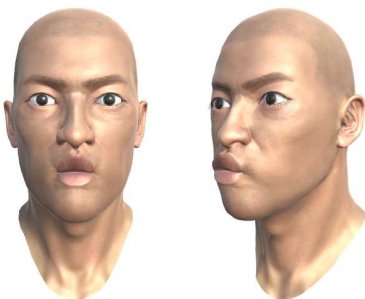
This blend shape pouts your lower lip.

Can be used with

[XR\\_LIP\\_EXPRESSION\\_MOUTH\\_UPPER\\_UPRIGHT\\_HTC](#) and

[XR\\_LIP\\_EXPRESSION\\_MOUTH\\_LOWER\\_DOWNRIGHT\\_HTC](#) to complete upper O mouth shape.

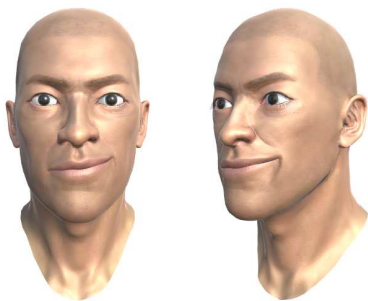
#### XR\_LIP\_EXPRESSION\_MOUTH\_POUT\_HTC



##### **Description**

This blend shape allows the lips to pout more with a higher value.

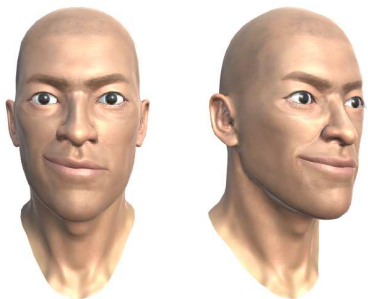
XR\_LIP\_EXPRESSION\_MOUTH\_SMILE\_LEFT\_HTC



**Description**

This blend shape raises the left side of the mouth further with a higher value.

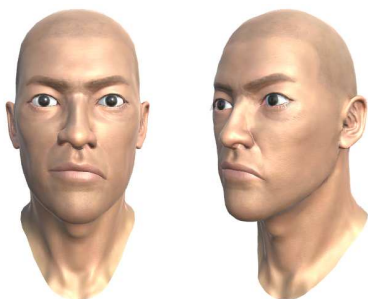
XR\_LIP\_EXPRESSION\_MOUTH\_SMILE\_RIGHT\_HTC



**Description**

This blend shape raises the right side of the mouth further with a higher value.

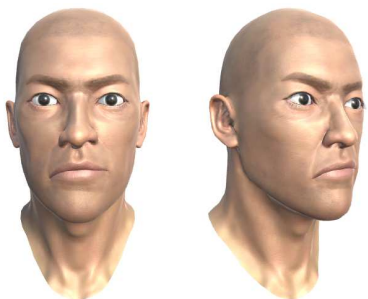
XR\_LIP\_EXPRESSION\_MOUTH\_SAD\_LEFT\_HTC



**Description**

This blend shape lowers the left side of the mouth further with a higher value.

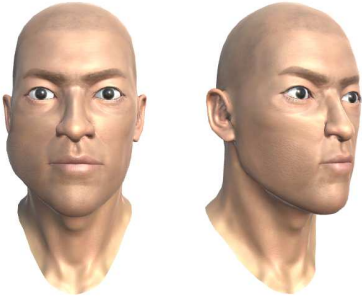
XR\_LIP\_EXPRESSION\_MOUTH\_SAD\_RIGHT\_HTC



**Description**

This blend shape lowers the right side of the mouth further with a higher value.

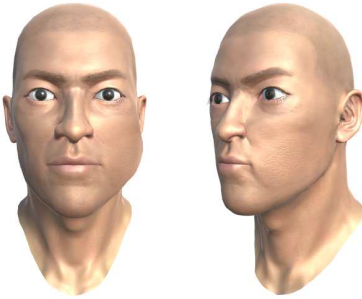
XR\_LIP\_EXPRESSION\_CHEEK\_PUFF\_RIGHT\_HTC



**Description**

This blend shape puffs up the right side of the cheek further with a higher value.

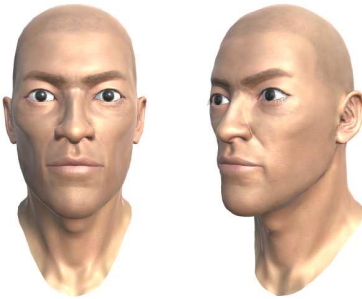
XR\_LIP\_EXPRESSION\_CHEEK\_PUFF\_LEFT\_HTC



**Description**

This blend shape puffs up the left side of the cheek further with a higher value.

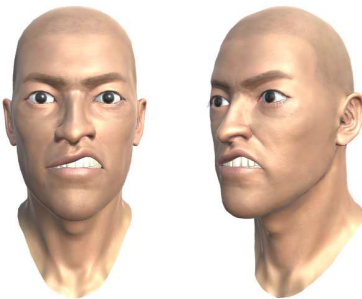
XR\_LIP\_EXPRESSION\_CHEEK\_SUCK\_HTC



**Description**

This blend shape sucks in the cheeks on both sides further with a higher value.

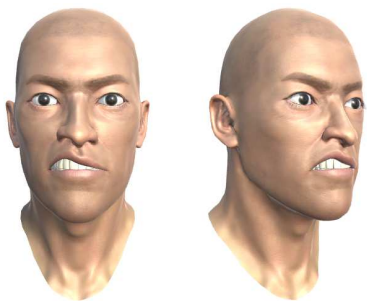
XR\_LIP\_EXPRESSION\_MOUTH\_UPPER\_UPLEFT\_HTC



**Description**

This blend shape raises the left upper lip further with a higher value.

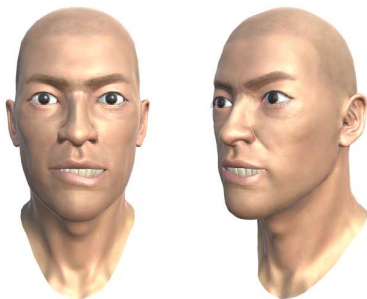
XR\_LIP\_EXPRESSION\_MOUTH\_UPPER\_UPRIGHT\_HTC



**Description**

This blend shape raises the right upper lip further with a higher value.

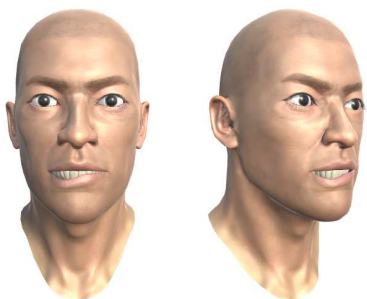
XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_DOWNLEFT\_HTC



**Description**

This blend shape lowers the left lower lip further with a higher value.

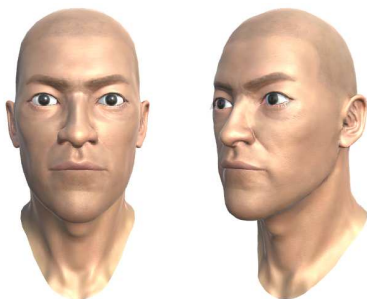
XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_DOWNRIGHT\_HTC



**Description**

This blend shape lowers the right lower lip further with a higher value.

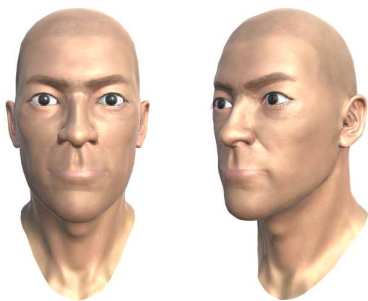
XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_INSIDE\_HTC



**Description**

This blend shape rolls in the lower lip further with a higher value.

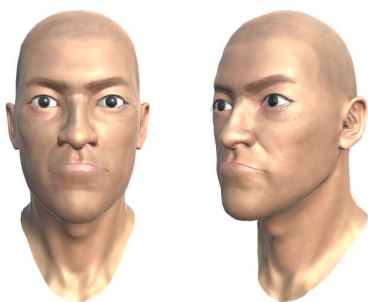
#### XR\_LIP\_EXPRESSION\_MOUTH\_UPPER\_INSIDE\_HTC



##### **Description**

This blend shape rolls in the upper lip further with a higher value.

#### XR\_LIP\_EXPRESSION\_MOUTH\_LOWER\_OVERLAY\_HTC



##### **Description**

This blend shape stretches the lower lip further and lays it on the upper lip further with a higher value.

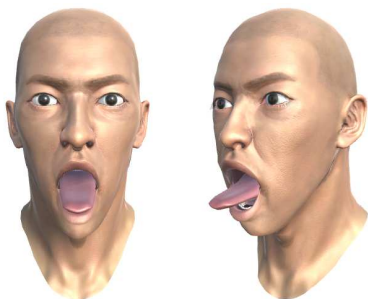
#### XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP1\_HTC



##### **Description**

This blend shape sticks the tongue out slightly.  
In step 1 of extending the tongue, the main action of the tongue is to lift up, and the elongated length only extends to a little bit beyond the teeth.

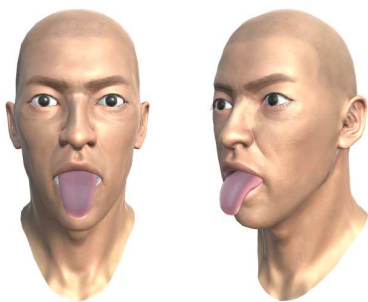
#### XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP2\_HTC



##### **Description**

This blend shape sticks the tongue out extremely.  
Continuing the step 1, it extends the tongue to the longest.

### XR\_LIP\_EXPRESSION\_TONGUE\_DOWN\_HTC



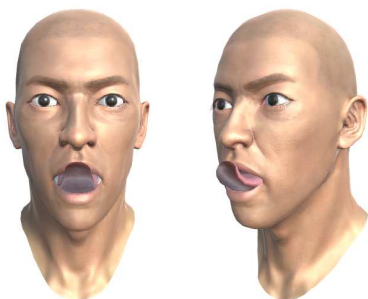
#### Description

This blend shape sticks the tongue out and down extremely.

This example contains

(XR\_LIP\_EXPRESSION\_TONGUE\_DOWN\_HTC  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP1\_HTC  
+  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP2\_HTC  
).

### XR\_LIP\_EXPRESSION\_TONGUE\_UP\_HTC



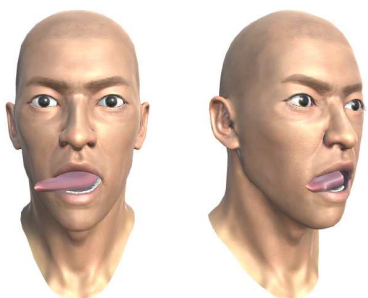
#### Description

This blend shape sticks the tongue out and up extremely.

This example contains

(XR\_LIP\_EXPRESSION\_TONGUE\_UP\_HTC  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP1\_HTC  
+  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP2\_HTC  
).

### XR\_LIP\_EXPRESSION\_TONGUE\_RIGHT\_HTC



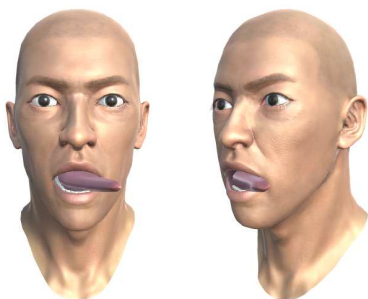
#### Description

This blend shape sticks the tongue out and right extremely.

This example contains

(XR\_LIP\_EXPRESSION\_TONGUE\_RIGHT\_HTC  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP1\_HTC  
+  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP2\_HTC  
).

### XR\_LIP\_EXPRESSION\_TONGUE\_LEFT\_HTC



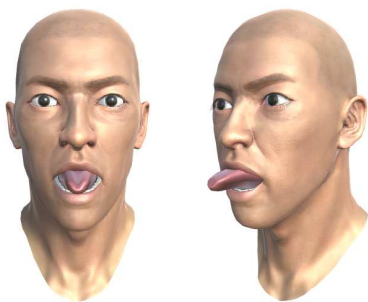
#### Description

This blend shape sticks the tongue out and left extremely.

This example contains

(XR\_LIP\_EXPRESSION\_TONGUE\_LEFT\_HTC  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP1\_HTC  
+  
XR\_LIP\_EXPRESSION\_TONGUE\_LONGSTEP2\_HTC  
).

## XR\_LIP\_EXPRESSION\_TONGUE\_ROLL\_HTC



### Description

This blend shape sticks the tongue out with roll type.

This example contains

```
(XR_LIP_EXPRESSION_TONGUE_ROLL_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
+  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
).
```

## XR\_LIP\_EXPRESSION\_TONGUE\_UPRIGHT\_MORPH\_HTC



### Description

This blend shape does not make sense. When both the right and up blend shapes appear at the same time, the tongue will be deformed.

```
(XR_LIP_EXPRESSION_TONGUE_RIGHT_HTC  
XR_LIP_EXPRESSION_TONGUE_UP_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
)
```



### Description

This blend shape fixes the deformation illustrated above.

```
(XR_LIP_EXPRESSION_TONGUE_RIGHT_HTC  
XR_LIP_EXPRESSION_TONGUE_UP_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
+  
XR_LIP_EXPRESSION_TONGUE_UPRIGHT_MORPH_HTC)
```

## XR\_LIP\_EXPRESSION\_TONGUE\_ULEFT\_MORPH\_HTC



### Description

This blend shape does not make sense. When both the left and up blend shapes appear at the same time, the tongue will be deformed.

```
(XR_LIP_EXPRESSION_TONGUE_LEFT_HTC  
XR_LIP_EXPRESSION_TONGUE_UP_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
)
```



### Description

This blend shape fixes the deformation illustrated above.

```
(XR_LIP_EXPRESSION_TONGUE_LEFT_HTC  
XR_LIP_EXPRESSION_TONGUE_UP_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
+  
XR_LIP_EXPRESSION_TONGUE_ULEFT_MORPH_HTC)
```

## XR\_LIP\_EXPRESSION\_TONGUE\_DOWNRIGHT\_MORPH\_HTC



### Description

This blend shape does not make sense. When both the right and down blend shapes appear at the same time, the tongue will be deformed.

```
(XR_LIP_EXPRESSION_TONGUE_RIGHT_HTC  
XR_LIP_EXPRESSION_TONGUE_DOWN_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
)
```



## XR\_LIP\_EXPRESSION\_TONGUE\_DOWNRIGHT\_MORPH\_HTC



### Description

This blend shape fixes the deformation illustrated above.

```
(XR_LIP_EXPRESSION_TONGUE_RIGHT_HTC  
XR_LIP_EXPRESSION_TONGUE_DOWN_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
+  
XR_LIP_EXPRESSION_TONGUE_DOWNRIGHT_MORPH_HTC)
```

## XR\_LIP\_EXPRESSION\_TONGUE\_DOWNLEFT\_MORPH\_HTC



### Description

This blend shape does not make sense. When both the left and down blend shapes appear at the same time, the tongue will be deformed.

```
(XR_LIP_EXPRESSION_TONGUE_LEFT_HTC  
XR_LIP_EXPRESSION_TONGUE_DOWN_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
)
```

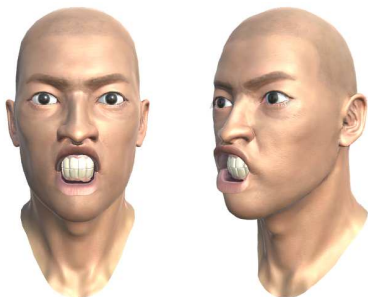


### Description

This blend shape fixes the deformation illustrated above.

```
(XR_LIP_EXPRESSION_TONGUE_LEFT_HTC  
XR_LIP_EXPRESSION_TONGUE_DOWN_HTC +  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP1_HTC  
XR_LIP_EXPRESSION_TONGUE_LONGSTEP2_HTC  
+  
XR_LIP_EXPRESSION_TONGUE_DOWNLEFT_MORPH_HTC)
```

## O shape

|   |  |
|---|--|
|  | <p><b>Description</b></p> <p>The entire O-shaped mouth is formed by the combination of 6 blend shapes: (XR_LIP_EXPRESSION_MOUTH_UPPER_OVERTURN_HTC<br/>XR_LIP_EXPRESSION_MOUTH_LOWER_OVERTURN_HTC<br/>XR_LIP_EXPRESSION_MOUTH_UPPER_UPLIFT_HTC<br/>XR_LIP_EXPRESSION_MOUTH_UPPER_UPRIGHT_HTC<br/>XR_LIP_EXPRESSION_MOUTH_LOWER_DOWNLEFT_HTC<br/>XR_LIP_EXPRESSION_MOUTH_LOWER_DOWNRIGHT_HTC)</p> |
|---|--|

### New Object Types

- [XrFacialTrackerHTC](#)

### New Flag Types

### New Enum Constants

[XrObjectType](#) enumeration is extended with:

- `XR_OBJECT_TYPE_FACIAL_TRACKER_HTC`

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SYSTEM_FACIAL_TRACKING_PROPERTIES_HTC`
- `XR_TYPE_FACIAL_TRACKER_CREATE_INFO_HTC`
- `XR_TYPE_FACIAL_EXPRESSIONS_HTC`

### New Enums

- [XrFacialTrackingTypeHTC](#)
- [XrEyeExpressionHTC](#)
- [XrLipExpressionHTC](#)

### New Structures

- [XrSystemFacialTrackingPropertiesHTC](#)
- [XrFacialTrackerCreateInfoHTC](#)

- [XrFacialExpressionsHTC](#)

### New Functions

- [xrCreateFacialTrackerHTC](#)
- [xrDestroyFacialTrackerHTC](#)
- [xrGetFacialExpressionsHTC](#)

### Issues

### Version History

- Revision 1, 2021-12-16 (Kyle Chen)
  - Initial extension description
- Revision 2, 2022-09-22 (Andy Chen)
  - Correct the range of the blink blend shapes.

## 12.85. XR\_HTC\_foveation

### Name String

`XR_HTC_foveation`

### Extension Type

Instance extension

### Registered Extension Number

319

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-09-14

### IP Status

No known IP claims.

### Contributors

Billy Chang, HTC  
Bill Chang, HTC

## Overview

This extension enables an application to gain rendering performance improvement by reducing the pixel density of areas in the peripheral vision. The areas near the focal point still sustains the original pixel density than periphery.

The application **can** use this extension in the following steps:

1. Create an [XrFoveationApplyInfoHTC](#) structure with the desired foveation configurations.
2. Apply the foveation configuration by calling [xrApplyFoveationHTC](#) with desired [XrFoveationApplyInfoHTC](#).



### Note

This extension is recommended for [XrSession](#) whose [XrViewConfigurationType](#) is [XR\\_VIEW\\_CONFIGURATION\\_TYPE\\_PRIMARY\\_STEREO](#).

## Operate foveated rendering

The application **can** operate foveated rendering by calling [xrApplyFoveationHTC](#) with the corresponding foveation configuration and the specified [XrSwapchainSubImage](#).

The [xrApplyFoveationHTC](#) function is defined as:

```
// Provided by XR_HTC_foveation
XrResult xrApplyFoveationHTC(
    XrSession session,
    const XrFoveationApplyInfoHTC* applyInfo);
```

## Parameter Descriptions

- [session](#) is a handle to an [XrSession](#) in which the foveation will apply to.
- [applyInfo](#) is a pointer to an [XrFoveationApplyInfoHTC](#) structure containing information about the foveation configuration and applied [XrSwapchainSubImage](#).

The foveation configuration will be applied after this call, and the state will persist until the next call to [xrApplyFoveationHTC](#) or the end of this [XrSession](#), whichever comes first. You **should** not call [xrApplyFoveationHTC](#) during rendering to target image layer [XrSwapchainSubImage](#) in render loop.

## Valid Usage (Implicit)

- The `XR_HTC_foveation` extension **must** be enabled prior to calling `xrApplyFoveationHTC`
- `session` **must** be a valid `XrSession` handle
- `applyInfo` **must** be a pointer to a valid `XrFoveationApplyInfoHTC` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_LIMIT_REACHED`

The `XrFoveationApplyInfoHTC` structure is defined as:

```
// Provided by XR_HTC_foveation
typedef struct XrFoveationApplyInfoHTC {
    XrStructureType      type;
    const void*          next;
    XrFoveationModeHTC  mode;
    uint32_t             subImageCount;
    XrSwapchainSubImage* subImages;
} XrFoveationApplyInfoHTC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `mode` is an [XrFoveationModeHTC](#) enum describing the foveation mode.
- `subImageCount` is the count of `subImages` in the `subImages` array. This **must** be equal to the number of view poses returned by [xrLocateViews](#).
- `subImages` is an array of [XrSwapchainSubImage](#) to apply foveated rendering.

The application **should** set the following configurations in [XrFoveationApplyInfoHTC](#):

- The foveation mode to be applied.
- The specified [XrSwapchainSubImage](#) to the corresponding view.

The [XrSwapchain::faceCount](#) of the swapchain in [XrSwapchainSubImage](#) **must** be 1 since this extension does not support cubemaps.

If `mode` is `XR_FOVEATION_MODE_DYNAMIC_HTC`, the `next` chain for this structure **must** include [XrFoveationDynamicModeInfoHTC](#) structure.

If `mode` is `XR_FOVEATION_MODE_CUSTOM_HTC`, the `next` chain for this structure **must** include [XrFoveationCustomModeInfoHTC](#) structure.

The order of `subImages` **must** be the same order as in [XrCompositionLayerProjectionView](#) when submitted in [xrEndFrame](#).

## Valid Usage (Implicit)

- The `XR_HTC_foveation` extension **must** be enabled prior to using [XrFoveationApplyInfoHTC](#)
- `type` **must** be `XR_TYPE_FOVEATION_APPLY_INFO_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrFoveationCustomModeInfoHTC](#), [XrFoveationDynamicModeInfoHTC](#)
- `mode` **must** be a valid [XrFoveationModeHTC](#) value
- `subImages` **must** be a pointer to an array of `subImageCount` [XrSwapchainSubImage](#) structures
- The `subImageCount` parameter **must** be greater than 0

[XrFoveationModeHTC](#) identifies the different foveation modes.

```
// Provided by XR_HTC_foveation
typedef enum XrFoveationModeHTC {
    XR_FOVEATION_MODE_DISABLE_HTC = 0,
    XR_FOVEATION_MODE_FIXED_HTC = 1,
    XR_FOVEATION_MODE_DYNAMIC_HTC = 2,
    XR_FOVEATION_MODE_CUSTOM_HTC = 3,
    XR_FOVEATION_MODE_MAX_ENUM_HTC = 0x7FFFFFFF
} XrFoveationModeHTC;
```

## Enumerant Descriptions

- `XR_FOVEATION_MODE_DISABLE_HTC` — No foveation
- `XR_FOVEATION_MODE_FIXED_HTC` — Apply system default setting with fixed clear FOV and periphery quality.
- `XR_FOVEATION_MODE_DYNAMIC_HTC` — Allow system to set foveation dynamically according realtime system metric or other extensions.
- `XR_FOVEATION_MODE_CUSTOM_HTC` — Allow application to set foveation with desired clear FOV, periphery quality, and focal center offset.

### Dynamic foveation mode

The application allows runtime to configure the foveation settings dynamically according to the system metrics or other extensions.

The `XrFoveationDynamicModeInfoHTC` structure is defined as:

```
// Provided by XR_HTC_foveation
typedef struct XrFoveationDynamicModeInfoHTC {
    XrStructureType          type;
    const void*              next;
    XrFoveationDynamicFlagsHTC dynamicFlags;
} XrFoveationDynamicModeInfoHTC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `dynamicFlags` is a bitmask of [XrFoveationDynamicFlagBitsHTC](#) indicated which item **may** be changed during dynamic mode.

The application **must** chain an [XrFoveationDynamicModeInfoHTC](#) structure to [XrFoveationApplyInfoHTC](#) if dynamic mode is set.

## Valid Usage (Implicit)

- The [XR\\_HTC\\_foveation](#) extension **must** be enabled prior to using [XrFoveationDynamicModeInfoHTC](#)
- `type` **must** be `XR_TYPE_FOVEATION_DYNAMIC_MODE_INFO_HTC`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `dynamicFlags` **must** be `0` or a valid combination of [XrFoveationDynamicFlagBitsHTC](#) values

```
typedef XrFlags64 XrFoveationDynamicFlagsHTC;
```

```
// Flag bits for XrFoveationDynamicFlagsHTC
static const XrFoveationDynamicFlagsHTC XR_FOVEATION_DYNAMIC_LEVEL_ENABLED_BIT_HTC =
0x00000001;
static const XrFoveationDynamicFlagsHTC XR_FOVEATION_DYNAMIC_CLEAR_FOV_ENABLED_BIT_HTC =
0x00000002;
static const XrFoveationDynamicFlagsHTC
XR_FOVEATION_DYNAMIC_FOCAL_CENTER_OFFSET_ENABLED_BIT_HTC = 0x00000004;
```



## Flag Descriptions

- `XR_FOVEATION_DYNAMIC_LEVEL_ENABLED_BIT_HTC` — Allow system to set periphery pixel density dynamically.
- `XR_FOVEATION_DYNAMIC_CLEAR_FOV_ENABLED_BIT_HTC` — Allow system to set clear FOV degree dynamically.
- `XR_FOVEATION_DYNAMIC_FOCAL_CENTER_OFFSET_ENABLED_BIT_HTC` — Allow system to set focal center offset dynamically.

### Custom foveation mode

The application **can** configure the foveation settings according to the preference of content.

The `XrFoveationCustomModeInfoHTC` structure is defined as:

```
// Provided by XR_HTC_foveation
typedef struct XrFoveationCustomModeInfoHTC {
    XrStructureType          type;
    const void*              next;
    uint32_t                  configCount;
    const XrFoveationConfigurationHTC* configs;
} XrFoveationCustomModeInfoHTC;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `configCount` is a `uint32_t` describing the count of elements in the `configs` array, which **must** be the number of views.
- `configs` is an array of `XrFoveationConfigurationHTC` structure contains the custom foveation settings for the corresponding views.

The application **must** chain an `XrFoveationCustomModeInfoHTC` structure to `XrFoveationApplyInfoHTC` to customize foveation if custom mode is set.

## Valid Usage (Implicit)

- The `XR_HTC_foveation` extension **must** be enabled prior to using `XrFoveationCustomModeInfoHTC`
- `type` **must** be `XR_TYPE_FOVEATION_CUSTOM_MODE_INFO_HTC`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `configs` **must** be a pointer to an array of `configCount` valid `XrFoveationConfigurationHTC` structures
- The `configCount` parameter **must** be greater than 0

The `XrFoveationConfigurationHTC` structure is defined as:

```
// Provided by XR_HTC_foveation
typedef struct XrFoveationConfigurationHTC {
    XrFoveationLevelHTC    level;
    float                  clearFovDegree;
    XrVector2f             focalCenterOffset;
} XrFoveationConfigurationHTC;
```

## Member Descriptions

- `level` is the pixel density drop level of periphery area specified by `XrFoveationLevelHTC`.
- `clearFovDegree` is the value indicating the total horizontal and vertical field angle with the original pixel density level. `clearFovDegree` **must** be specified in degree, and **must** be in the range [0, 180].
- `focalCenterOffset` is the desired center offset of the field of view in NDC(normalized device coordinates) space. The x and y of `focalCenterOffset` **must** be in the range [-1, 1].

## Valid Usage (Implicit)

- The `XR_HTC_foveation` extension **must** be enabled prior to using `XrFoveationConfigurationHTC`
- `level` **must** be a valid `XrFoveationLevelHTC` value

```
// Provided by XR_HTC_foveation
typedef enum XrFoveationLevelHTC {
    XR_FOVEATION_LEVEL_NONE_HTC = 0,
    XR_FOVEATION_LEVEL_LOW_HTC = 1,
    XR_FOVEATION_LEVEL_MEDIUM_HTC = 2,
    XR_FOVEATION_LEVEL_HIGH_HTC = 3,
    XR_FOVEATION_LEVEL_MAX_ENUM_HTC = 0x7FFFFFFF
} XrFoveationLevelHTC;
```

## Enumerant Descriptions

- `XR_FOVEATION_LEVEL_NONE_HTC` — No foveation
- `XR_FOVEATION_LEVEL_LOW_HTC` — Light periphery pixel density drop and lower performance gain.
- `XR_FOVEATION_LEVEL_MEDIUM_HTC` — Medium periphery pixel density drop and medium performance gain
- `XR_FOVEATION_LEVEL_HIGH_HTC` — Heavy periphery pixel density drop and higher performance gain

### New Object Types

### New Flag Types

[XrFoveationDynamicFlagsHTC](#)

### New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_FOVEATION_APPLY_INFO_HTC`
- `XR_TYPE_FOVEATION_DYNAMIC_MODE_INFO_HTC`
- `XR_TYPE_FOVEATION_CUSTOM_MODE_INFO_HTC`

### New Enum Constants

### New Enums

[XrFoveationModeHTC](#)

[XrFoveationDynamicFlagBitsHTC](#)

[XrFoveationLevelHTC](#)

## New Structures

[XrFoveationApplyInfoHTC](#)

[XrFoveationDynamicModeInfoHTC](#)

[XrFoveationCustomModeInfoHTC](#)

## New Functions

[xrApplyFoveationHTC](#)

## Issues

## Version History

- Revision 1, 2022-09-14 (Billy Chang)
  - Initial extension description

# 12.86. XR\_HTC\_hand\_interaction

## Name String

`XR_HTC_hand_interaction`

## Extension Type

Instance extension

## Registered Extension Number

107

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## API Interactions

- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

## Last Modified Date

2022-05-27

## IP Status

No known IP claims.

## Contributors

Ria Hsu, HTC  
Bill Chang, HTC

## Overview

This extension defines a new interaction profile for tracked hands.

## Hand interaction profile

Interaction profile path:

- */interaction\_profiles/htc/hand\_interaction*

### Note

The interaction profile path */interaction\_profiles/htc/hand\_interaction* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/htc/hand\_interaction\_htc*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths:

- */user/hand\_htc/left*
- */user/hand\_htc/right*

This interaction profile represents basic pose and actions for interaction of tracked hands.

Supported component paths for far interaction:

- *.../input/select/value*
- *.../input/aim/pose*

The application **should** use *.../input/aim/pose* path to aim at objects in the world and use *.../input/select/value* path to decide user selection from pinch shape strength which the range of value is *0.0f* to *1.0f*, with *1.0f* meaning pinch fingers touched.

Supported component paths for near interaction:

- *.../input/squeeze/value*
- *.../input/grip/pose*

The application **should** use *.../input/grip/pose* path to interact with the nearby objects and locate the position of handheld objects, and use *.../input/squeeze/value* path to decide the hand picking up or holding the nearby objects from grip shape strength which the range of value is *0.0f* to *1.0f*, with *1.0f*

meaning hand grip shape is closed.



*Note*

Far and near interaction depends on the support capabilities of hand tracking engine. The application **can** check `isActive` of `XrActionStatePose` of aim and grip to know far and near interaction supported or not then decide the interaction behavior in content.

## Version History

- Revision 1, 2022-05-27 (Ria Hsu)
  - Initial extension description

# 12.87. XR\_HTC\_passthrough

## Name String

`XR_HTC_passthrough`

## Extension Type

Instance extension

## Registered Extension Number

318

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-09-14

## IP Status

No known IP claims.

## Contributors

Livi Lin, HTC

Sacdar Hsu, HTC

Bill Chang, HTC

## Overview

This extension enables an application to show the passthrough image to see the surrounding environment from the VR headset. The application is allowed to configure the passthrough image with

the different appearances according to the demand of the application.

The passthrough configurations that runtime provides to applications contain:

- Decide the passthrough layer shown over or under the frame submitted by the application.
- Specify the passthrough form with full of the entire screen or projection onto the mesh specified by the application.
- Set the alpha blending level for the composition of the passthrough layer.

### Create a passthrough handle

An application **can** create an [XrPassthroughHTC](#) handle by calling [xrCreatePassthroughHTC](#). The returned passthrough handle **can** be subsequently used in API calls.

```
// Provided by XR_HTC_passthrough
XR_DEFINE_HANDLE(XrPassthroughHTC)
```

The [xrCreatePassthroughHTC](#) function is defined as:

```
// Provided by XR_HTC_passthrough
XrResult xrCreatePassthroughHTC(
    XrSession session,
    const XrPassthroughCreateInfoHTC* createInfo,
    XrPassthroughHTC* passthrough);
```

### Parameter Descriptions

- **session** is an [XrSession](#) in which the passthrough will be active.
- **createInfo** is a pointer to an [XrPassthroughCreateInfoHTC](#) structure containing information about how to create the passthrough.
- **passthrough** is a pointer to a handle in which the created [XrPassthroughHTC](#) is returned.

Creates an [XrPassthroughHTC](#) handle.

If the function successfully returned, the output **passthrough** **must** be a valid handle.

## Valid Usage (Implicit)

- The `XR_HTC_passthrough` extension **must** be enabled prior to calling `xrCreatePassthroughHTC`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrPassthroughCreateInfoHTC` structure
- `passthrough` **must** be a pointer to an `XrPassthroughHTC` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrPassthroughCreateInfoHTC` structure is defined as:

```
// Provided by XR_HTC_passthrough
typedef struct XrPassthroughCreateInfoHTC {
    XrStructureType      type;
    const void*          next;
    XrPassthroughFormHTC form;
} XrPassthroughCreateInfoHTC;
```



## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is NULL or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **form** [XrPassthroughFormHTC](#) that specifies the form of passthrough.

## Valid Usage (Implicit)

- The [XR\\_HTC\\_passthrough](#) extension **must** be enabled prior to using [XrPassthroughCreateInfoHTC](#)
- **type** **must** be `XR_TYPE_PASSTHROUGH_CREATE_INFO_HTC`
- **next** **must** be NULL or a valid pointer to the [next structure in a structure chain](#)
- **form** **must** be a valid [XrPassthroughFormHTC](#) value

The [XrPassthroughFormHTC](#) enumeration identifies the form of the passthrough, presenting the passthrough fill the full screen or project onto a specified mesh.

```
// Provided by XR_HTC_passthrough
typedef enum XrPassthroughFormHTC {
    XR_PASSTHROUGH_FORM_PLANAR_HTC = 0,
    XR_PASSTHROUGH_FORM_PROJECTED_HTC = 1,
    XR_PASSTHROUGH_FORM_MAX_ENUM_HTC = 0x7FFFFFFF
} XrPassthroughFormHTC;
```

## Enumerant Descriptions

- `XR_PASSTHROUGH_FORM_PLANAR_HTC` — Presents the passthrough with full of the entire screen.
- `XR_PASSTHROUGH_FORM_PROJECTED_HTC` — Presents the passthrough projecting onto a custom mesh.

The [xrDestroyPassthroughHTC](#) function is defined as:

```
// Provided by XR_HTC_passthrough
XrResult xrDestroyPassthroughHTC(
    XrPassthroughHTC                passthrough);
```

## Parameter Descriptions

- `passthrough` is the `XrPassthroughHTC` to be destroyed.

The `xrDestroyPassthroughHTC` function releases the passthrough and the underlying resources.

## Valid Usage (Implicit)

- The `XR_HTC_passthrough` extension **must** be enabled prior to calling `xrDestroyPassthroughHTC`
- `passthrough` **must** be a valid `XrPassthroughHTC` handle

## Thread Safety

- Access to `passthrough`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`

## Composite the passthrough layer

The `XrCompositionLayerPassthroughHTC` structure is defined as:

```
// Provided by XR_HTC_passthrough
typedef struct XrCompositionLayerPassthroughHTC {
    XrStructureType      type;
    const void*          next;
    XrCompositionLayerFlags  layerFlags;
    XrSpace              space;
    XrPassthroughHTC     passthrough;
    XrPassthroughColorHTC color;
} XrCompositionLayerPassthroughHTC;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain, such as [XrPassthroughMeshTransformInfoHTC](#).
- `layerFlags` is a bitmask of [XrCompositionLayerFlagBits](#) describing flags to apply to the layer.
- `space` is the [XrSpace](#) that specifies the layer's space - **must** be [XR\\_NULL\\_HANDLE](#).
- `passthrough` is the [XrPassthroughHTC](#) previously created by [xrCreatePassthroughHTC](#).
- `color` is the [XrPassthroughColorHTC](#) describing the color information with the alpha value of the passthrough layer.

The application **can** create an [XrCompositionLayerPassthroughHTC](#) structure with the created `passthrough` and the corresponding information. A pointer to [XrCompositionLayerPassthroughHTC](#) **may** be submitted in [xrEndFrame](#) as a pointer to the base structure [XrCompositionLayerBaseHeader](#), in the desired layer order, to request the runtime to composite a passthrough layer into the final frame output.

If the passthrough form specified to [xrCreatePassthroughHTC](#) is [XR\\_PASSTHROUGH\\_FORM\\_PROJECTED\\_HTC](#), [XrPassthroughMeshTransformInfoHTC](#) **must** appear in the `next` chain. If they are absent, the runtime **must** return error [XR\\_ERROR\\_VALIDATION\\_FAILURE](#).

## Valid Usage (Implicit)

- The `XR_HTC_passthrough` extension **must** be enabled prior to using `XrCompositionLayerPassthroughHTC`
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_PASSTHROUGH_HTC`
- `next` **must** be `NULL` or a valid pointer to the next structure in a structure chain. See also: `XrPassthroughMeshTransformInfoHTC`
- `layerFlags` **must** be a valid combination of `XrCompositionLayerFlagBits` values
- `layerFlags` **must** not be `0`
- `space` **must** be a valid `XrSpace` handle
- `passthrough` **must** be a valid `XrPassthroughHTC` handle
- `color` **must** be a valid `XrPassthroughColorHTC` structure
- Both of `passthrough` and `space` **must** have been created, allocated, or retrieved from the same `XrSession`

The `XrPassthroughColorHTC` structure is defined as:

```
// Provided by XR_HTC_passthrough
typedef struct XrPassthroughColorHTC {
    XrStructureType    type;
    const void*        next;
    float               alpha;
} XrPassthroughColorHTC;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `alpha` is the alpha value of the passthrough in the range [0, 1].

The application **can** specify the `XrPassthroughColorHTC` to adjust the alpha value of the passthrough. The range is between 0.0f and 1.0f, 1.0f means opaque.

## Valid Usage (Implicit)

- The `XR_HTC_passthrough` extension **must** be enabled prior to using `XrPassthroughColorHTC`
- `type` **must** be `XR_TYPE_PASSTHROUGH_COLOR_HTC`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

The `XrPassthroughMeshTransformInfoHTC` structure is defined as:

```
// Provided by XR_HTC_passthrough
typedef struct XrPassthroughMeshTransformInfoHTC {
    XrStructureType    type;
    const void*        next;
    uint32_t           vertexCount;
    const XrVector3f*  vertices;
    uint32_t           indexCount;
    const uint32_t*    indices;
    XrSpace            baseSpace;
    XrTime             time;
    XrPosef            pose;
    XrVector3f         scale;
} XrPassthroughMeshTransformInfoHTC;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `vertexCount` is the count of vertices array in the mesh.
- `vertices` is an array of `XrVector3f`. The size of the array **must** be equal to `vertexCount`.
- `indexCount` is the count of indices array in the mesh.
- `indices` is an array of triangle indices. The size of the array **must** be equal to `indexCount`.
- `baseSpace` is the `XrSpace` that defines the projected passthrough's base space for transformations.
- `time` is the `XrTime` that defines the time at which the transform is applied.
- `pose` is the `XrPosef` that defines the pose of the mesh
- `scale` is the `XrVector3f` that defines the scale of the mesh

The `XrPassthroughMeshTransformInfoHTC` structure describes the mesh and transformation.

The application **must** specify the `XrPassthroughMeshTransformInfoHTC` in the `next` chain of `XrCompositionLayerPassthroughHTC` if the specified form of passthrough layer previously created by `xrCreatePassthroughHTC` is `XR_PASSTHROUGH_FORM_PROJECTED_HTC`.

Passing `XrPassthroughMeshTransformInfoHTC` updates the projected mesh information in the runtime for passthrough layer composition.

If `XrPassthroughMeshTransformInfoHTC` is not set correctly, runtime **must** return error `XR_ERROR_VALIDATION_FAILURE` when `xrEndFrame` is called with composition layer `XrCompositionLayerPassthroughHTC`.

### Valid Usage (Implicit)

- The `XR_HTC_passthrough` extension **must** be enabled prior to using `XrPassthroughMeshTransformInfoHTC`
- `type` **must** be `XR_TYPE_PASSTHROUGH_MESH_TRANSFORM_INFO_HTC`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `vertices` **must** be a pointer to an array of `vertexCount` `XrVector3f` structures
- `indices` **must** be a pointer to an array of `indexCount` `uint32_t` values
- `baseSpace` **must** be a valid `XrSpace` handle
- The `vertexCount` parameter **must** be greater than 0
- The `indexCount` parameter **must** be greater than 0

### New Object Types

- `XrPassthroughHTC`

### New Flag Types

### New Enum Constants

`XrObjectType` enumeration is extended with:

- `XR_OBJECT_TYPE_PASSTHROUGH_HTC`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_PASSTHROUGH_CREATE_INFO_HTC`
- `XR_TYPE_PASSTHROUGH_COLOR_HTC`
- `XR_TYPE_PASSTHROUGH_MESH_TRANSFORM_INFO_HTC`

- [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_PASSTHROUGH\\_HTC](#)

### New Enums

- [XrPassthroughFormHTC](#)

### New Structures

- [XrPassthroughCreateInfoHTC](#)
- [XrPassthroughColorHTC](#)
- [XrPassthroughMeshTransformInfoHTC](#)
- [XrCompositionLayerPassthroughHTC](#)

### New Functions

- [xrCreatePassthroughHTC](#)
- [xrDestroyPassthroughHTC](#)

### Issues

### Version History

- Revision 1, 2022-09-14 (Sacdar Hsu)
  - Initial extension description

## 12.88. XR\_HTC\_vive\_wrist\_tracker\_interaction

### Name String

[XR\\_HTC\\_vive\\_wrist\\_tracker\\_interaction](#)

### Extension Type

Instance extension

### Registered Extension Number

108

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-05-27

## IP Status

No known IP claims.

## Contributors

Ria Hsu, HTC

Bill Chang, HTC

## Overview

This extension provides an [XrPath](#) for getting device input from a VIVE wrist tracker to enable its interactions. VIVE wrist tracker is a tracked device mainly worn on user's wrist for pose tracking. Besides this use case, user also can tie it to a physical object to track its object pose, e.g. tie on a gun.

## VIVE Wrist Tracker input

This extension exposes a new interaction profile path `/interaction_profiles/htc/vive_wrist_tracker` that is valid for the user path

- `/user/wrist_htc/left`
- `/user/wrist_htc/right`

for supported input source

- On `/user/wrist_htc/left` only:
  - `.../input/menu/click`
  - `.../input/x/click`
- On `/user/wrist_htc/right` only:
  - `.../input/system/click` (**may** not be available for application use)
  - `.../input/a/click`
- `.../input/entity_htc/pose`

### Note

The interaction profile path `/interaction_profiles/htc/vive_wrist_tracker` defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called `/interaction_profiles/htc/vive_wrist_tracker_htc`, to allow for modifications when promoted to a KHR extension or the core specification.

The `entity_htc` pose allows the applications to recognize the origin of a tracked input device, especially for the wearable devices which are not held in the user's hand. The `entity_htc` pose is defined as follows:

- The entity position: The center position of the tracked device.



- The entity orientation: Oriented with +Y up, +X to the right, and -Z forward.

## Version History

- Revision 1, 2022-05-27 (Ria Hsu)
  - Initial extension description

# 12.89. XR\_HUAWEI\_controller\_interaction

## Name String

`XR_HUAWEI_controller_interaction`

## Extension Type

Instance extension

## Registered Extension Number

70

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

## Last Modified Date

2020-05-26

## IP Status

No known IP claims.

## Contributors

Guodong Chen, Huawei

Kai Shao, Huawei

Yang Tao, Huawei

Gang Shen, Huawei

Yihong Huang, Huawei

## Overview

This extension defines a new interaction profile for the Huawei Controller, including but not limited to Huawei VR Glasses Controller.

## Huawei Controller interaction profile

Interaction profile path:

- */interaction\_profiles/huawei/controller*

### Note

The interaction profile path */interaction\_profiles/huawei/controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/huawei/controller\_huawei*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the Huawei Controller.

Supported component paths:

- *.../input/home/click*
- *.../input/back/click*
- *.../input/volume\_up/click*
- *.../input/volume\_down/click*
- *.../input/trigger/value*
- *.../input/trigger/click*
- *.../input/trackpad/x*
- *.../input/trackpad/y*
- *.../input/trackpad/click*
- *.../input/trackpad/touch*
- *.../input/aim/pose*
- *.../input/grip/pose*
- *.../output/haptic*



*Note*

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`



*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2020-04-28 (Yihong Huang)
  - Initial extension description

# 12.90. XR\_META\_automatic\_layer\_filter

## Name String

`XR_META_automatic_layer_filter`

## Extension Type

Instance extension

## Registered Extension Number

272

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_FB\\_composition\\_layer\\_settings](#)

## Contributors

Rohit Rao Padebettu, Meta  
Grant Yang, Meta

## Overview

This extension defines a new flag in [XrCompositionLayerSettingsFlagBitsFB](#) that allows applications to provide a hint to the runtime to automatically toggle a layer filtering mechanism. The layer filtering helps alleviate visual quality artifacts such as blur and flicker.

Note: The runtime **may** use any factors it wishes to apply a filter to the layer. These **may** include not only fixed factors such as screen resolution, HMD type, and swapchain resolution, but also dynamic ones such as layer pose and system-wide GPU utilization.

## Automatic Layer Filtering

[XrCompositionLayerSettingsFlagBitsFB](#) is extended with `XR_COMPOSITION_LAYER_SETTINGS_AUTO_LAYER_FILTER_BIT_META`

To enable automatic selection of layer filtering method, `XR_COMPOSITION_LAYER_SETTINGS_AUTO_LAYER_FILTER_BIT_META` is passed to the runtime in [XrCompositionLayerSettingsFB::layerFlags](#).

A candidate pool of preferred layer filtering methods from [XrCompositionLayerSettingsFlagBitsFB](#) **must** be passed along with `XR_COMPOSITION_LAYER_SETTINGS_AUTO_LAYER_FILTER_BIT_META`. The runtime **may** apply the appropriate filter when rendering the layer. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` from [xrEndFrame](#) when an [XrCompositionLayerSettingsFB](#) structure is

submitted with one or more of the layers if no other flag bits are supplied with [XR\\_COMPOSITION\\_LAYER\\_SETTINGS\\_AUTO\\_LAYER\\_FILTER\\_BIT\\_META](#).

## Version History

- Revision 1, 2023-04-21 (Rohit Rao Padebettu)
  - Initial extension description

# 12.91. XR\_META\_environment\_depth

## Name String

[XR\\_META\\_environment\\_depth](#)

## Extension Type

Instance extension

## Registered Extension Number

292

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2023-10-09

## IP Status

No known IP claims.

## Contributors

Andreas Selvik, Meta Platforms

Cass Everitt, Meta Platforms

Daniel Henell, Meta Platforms

John Kearney, Meta Platforms

Urs Niesen, Meta Platforms

### 12.91.1. Overview

This extension allows the application to request depth maps of the real-world environment around the headset. The depth maps are generated by the runtime and shared with the application using an [XrEnvironmentDepthSwapchainMETA](#).

## 12.91.2. Inspect System Capability

The `XrSystemEnvironmentDepthPropertiesMETA` structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrSystemEnvironmentDepthPropertiesMETA {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsEnvironmentDepth;
    XrBool32           supportsHandRemoval;
} XrSystemEnvironmentDepthPropertiesMETA;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsEnvironmentDepth` is an `XrBool32` indicating if current system supports environment depth.
- `supportsHandRemoval` is an `XrBool32` indicating if current system supports hand removal.

An application **can** inspect whether the system is capable of supporting environment depth by extending the `XrSystemProperties` with `XrSystemEnvironmentDepthPropertiesMETA` structure when calling `xrGetSystemProperties`.

If and only if a runtime returns `XR_FALSE` for `supportsEnvironmentDepth`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrCreateEnvironmentDepthProviderMETA`.

If and only if a runtime returns `XR_FALSE` for `supportsHandRemoval`, the runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from `xrSetEnvironmentDepthHandRemovalMETA`.

### Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to using `XrSystemEnvironmentDepthPropertiesMETA`
- `type` **must** be `XR_TYPE_SYSTEM_ENVIRONMENT_DEPTH_PROPERTIES_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.91.3. Creating and Destroying a Depth Provider

```
// Provided by XR_META_environment_depth
XR_DEFINE_HANDLE(XrEnvironmentDepthProviderMETA)
```

An [XrEnvironmentDepthProviderMETA](#) is a handle to a depth provider.

The [xrCreateEnvironmentDepthProviderMETA](#) function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrCreateEnvironmentDepthProviderMETA(
    XrSession session,
    const XrEnvironmentDepthProviderCreateInfoMETA* createInfo,
    XrEnvironmentDepthProviderMETA* environmentDepthProvider);
```

#### Parameter Descriptions

- `session` is the [XrSession](#).
- `createInfo` is a pointer to an [XrEnvironmentDepthProviderCreateInfoMETA](#) containing creation options for the depth provider.
- `environmentDepthProvider` is the returned [XrEnvironmentDepthProviderMETA](#) handle for the created depth provider.

The [xrCreateEnvironmentDepthProviderMETA](#) function creates a depth provider instance.

Creating the depth provider **may** allocate resources, but **should** not incur any per-frame compute costs until the provider has been started.

- Runtimes **must** create the provider in a stopped state.
- Runtimes **may** limit the number of depth providers per [XrInstance](#). If [xrCreateEnvironmentDepthProviderMETA](#) fails due to reaching this limit, the runtime **must** return `XR_ERROR_LIMIT_REACHED`.
- Runtimes **must** support at least 1 provider per [XrInstance](#).
- Runtimes **may** return `XR_ERROR_NOT_PERMITTED_PASSTHROUGH_FB` if the app permissions have not been granted to the calling app.
- Applications **can** call [xrStartEnvironmentDepthProviderMETA](#) to start the generation of depth maps.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling `xrCreateEnvironmentDepthProviderMETA`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrEnvironmentDepthProviderCreateInfoMETA` structure
- `environmentDepthProvider` **must** be a pointer to an `XrEnvironmentDepthProviderMETA` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_NOT_PERMITTED_PASSTHROUGH_FB`

The `XrEnvironmentDepthProviderCreateInfoMETA` structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthProviderCreateInfoMETA {
    XrStructureType                type;
    const void*                    next;
    XrEnvironmentDepthProviderCreateFlagsMETA createFlags;
} XrEnvironmentDepthProviderCreateInfoMETA;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `createFlags` is `0` or one or more [XrEnvironmentDepthProviderCreateFlagBitsMETA](#).

The [XrEnvironmentDepthProviderCreateInfoMETA](#) structure provides creation options for the [XrEnvironmentDepthProviderMETA](#) when passed to [xrCreateEnvironmentDepthProviderMETA](#).

## Valid Usage (Implicit)

- The [XR\\_META\\_environment\\_depth](#) extension **must** be enabled prior to using [XrEnvironmentDepthProviderCreateInfoMETA](#)
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_PROVIDER_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `createFlags` **must** be `0`

The [XrEnvironmentDepthProviderCreateFlagsMETA](#) specifies creation options for [XrEnvironmentDepthProviderMETA](#).

```
// Provided by XR_META_environment_depth
typedef XrFlags64 XrEnvironmentDepthProviderCreateFlagsMETA;
```

Valid bits for [XrEnvironmentDepthProviderCreateFlagsMETA](#) are defined by [XrEnvironmentDepthProviderCreateFlagBitsMETA](#), which is specified as:

```
// Provided by XR_META_environment_depth
// Flag bits for XrEnvironmentDepthProviderCreateFlagsMETA
```

There are currently no flag bits defined. This is reserved for future use.

The [xrDestroyEnvironmentDepthProviderMETA](#) function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrDestroyEnvironmentDepthProviderMETA(
    XrEnvironmentDepthProviderMETA          environmentDepthProvider);
```

## Parameter Descriptions

- `environmentDepthProvider` is an `XrEnvironmentDepthProviderMETA` handle for the depth provider.

The `xrDestroyEnvironmentDepthProviderMETA` function destroys the depth provider. After this call the runtime **may** free all related memory and resources.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling `xrDestroyEnvironmentDepthProviderMETA`
- `environmentDepthProvider` **must** be a valid `XrEnvironmentDepthProviderMETA` handle

## Thread Safety

- Access to `environmentDepthProvider`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`

## 12.91.4. Starting and Stopping a Depth Provider

The `xrStartEnvironmentDepthProviderMETA` function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrStartEnvironmentDepthProviderMETA(
    XrEnvironmentDepthProviderMETA          environmentDepthProvider);
```

## Parameter Descriptions

- `environmentDepthProvider` is an `XrEnvironmentDepthProviderMETA` handle for the depth provider.

The `xrStartEnvironmentDepthProviderMETA` function starts the asynchronous generation of depth maps.

Starting the depth provider **may** use CPU and GPU resources.

Runtimes **must** return `XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB` if `xrStartEnvironmentDepthProviderMETA` is called on an already started `XrEnvironmentDepthProviderMETA`.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling `xrStartEnvironmentDepthProviderMETA`
- `environmentDepthProvider` **must** be a valid `XrEnvironmentDepthProviderMETA` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB`

The [xrStopEnvironmentDepthProviderMETA](#) function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrStopEnvironmentDepthProviderMETA(
    XrEnvironmentDepthProviderMETA    environmentDepthProvider);
```

### Parameter Descriptions

- `environmentDepthProvider` is an [XrEnvironmentDepthProviderMETA](#) handle for the depth provider.

The [xrStopEnvironmentDepthProviderMETA](#) function stops the generation of depth maps. This stops all per frame computation of environment depth for the application.

Runtimes **must** return `XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB` if [xrStopEnvironmentDepthProviderMETA](#) is called on an already stopped [XrEnvironmentDepthProviderMETA](#).

### Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling [xrStopEnvironmentDepthProviderMETA](#)
- `environmentDepthProvider` **must** be a valid [XrEnvironmentDepthProviderMETA](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_UNEXPECTED_STATE_PASSTHROUGH_FB`

### 12.91.5. Hand Removal

Runtimes **may** provide functionality to remove hands from the depth map and filling in estimated background depth values. This is useful to support other occlusion methods specialized for hands to coexist with the Environment Depth extension.

The `xrSetEnvironmentDepthHandRemovalMETA` function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrSetEnvironmentDepthHandRemovalMETA(
    XrEnvironmentDepthProviderMETA    environmentDepthProvider,
    const XrEnvironmentDepthHandRemovalSetInfoMETA* setInfo);
```

### Parameter Descriptions

- `environmentDepthProvider` is an `XrEnvironmentDepthProviderMETA` handle for the depth provider.
- `setInfo` is a pointer to an `XrEnvironmentDepthHandRemovalSetInfoMETA` containing options for the hand removal.

The `xrSetEnvironmentDepthHandRemovalMETA` function sets hand removal options.

Runtimes **should** enable or disable the removal of the hand depths from the depth map. If enabled, the

corresponding depth pixels **should** be replaced with the estimated background depth behind the hands. Runtimes **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if and only if `XrSystemEnvironmentDepthPropertiesMETA::supportsHandRemoval` is `XR_FALSE`.

### Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling `xrSetEnvironmentDepthHandRemovalMETA`
- `environmentDepthProvider` **must** be a valid `XrEnvironmentDepthProviderMETA` handle
- `setInfo` **must** be a pointer to a valid `XrEnvironmentDepthHandRemovalSetInfoMETA` structure

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrEnvironmentDepthHandRemovalSetInfoMETA` structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthHandRemovalSetInfoMETA {
    XrStructureType    type;
    const void*        next;
    XrBool32           enabled;
} XrEnvironmentDepthHandRemovalSetInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `enabled` is `XR_TRUE` or `XR_FALSE` to enable/disable hand removal from the depth map, respectively.

This structure contains options passed to [xrSetEnvironmentDepthHandRemovalMETA](#).

## Valid Usage (Implicit)

- The [XR\\_META\\_environment\\_depth](#) extension **must** be enabled prior to using [XrEnvironmentDepthHandRemovalSetInfoMETA](#)
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_HAND_REMOVAL_SET_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.91.6. Creating a Readable Depth Swapchain

The depth data is generated in the runtime and shared to the application through an [XrEnvironmentDepthSwapchainMETA](#). This swapchain is different from regular swapchains in that it provides a data channel from the runtime to the application instead of the other way around.

```
// Provided by XR_META_environment_depth
XR_DEFINE_HANDLE(XrEnvironmentDepthSwapchainMETA)
```

[XrEnvironmentDepthSwapchainMETA](#) is a handle to a readable depth swapchain.

The [xrCreateEnvironmentDepthSwapchainMETA](#) function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrCreateEnvironmentDepthSwapchainMETA(
    XrEnvironmentDepthProviderMETA          environmentDepthProvider,
    const XrEnvironmentDepthSwapchainCreateInfoMETA* createInfo,
    XrEnvironmentDepthSwapchainMETA*       swapchain);
```

## Parameter Descriptions

- `environmentDepthProvider` is an [XrEnvironmentDepthProviderMETA](#) handle for the depth provider.
- `createInfo` is a pointer to an [XrEnvironmentDepthSwapchainCreateInfoMETA](#) containing creation options for the swapchain.
- `swapchain` is the returned [XrEnvironmentDepthSwapchainMETA](#) handle for the created swapchain.

The [xrCreateEnvironmentDepthSwapchainMETA](#) function creates a readable swapchain, which is used for accessing the depth data.

The runtime decides on the resolution and length of the swapchain. Additional information about the swapchain **can** be accessed by calling [xrGetEnvironmentDepthSwapchainStateMETA](#).

Runtimes **must** create a swapchain with array textures of length 2, which map to a left-eye and right-eye view. View index 0 **must** represent the left eye and view index 1 **must** represent the right eye. This is the same convention as for `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO` in [XrViewConfigurationType](#). Runtimes **must** create the swapchain with the following image formats depending on the graphics API associated with the session:

- OpenGL: `GL_DEPTH_COMPONENT16`
- Vulkan: `VK_FORMAT_D16_UNORM`
- Direct3D: `DXGI_FORMAT_D16_UNORM`

Runtimes **must** only allow maximum one swapchain to exist per depth provider at any given time, and must return `XR_ERROR_LIMIT_REACHED` if [xrCreateEnvironmentDepthSwapchainMETA](#) is called to create more. Applications **should** destroy the swapchain when no longer needed. Applications **must** be able to handle different swapchain lengths and resolutions.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling [xrCreateEnvironmentDepthSwapchainMETA](#)
- `environmentDepthProvider` **must** be a valid [XrEnvironmentDepthProviderMETA](#) handle
- `createInfo` **must** be a pointer to a valid [XrEnvironmentDepthSwapchainCreateInfoMETA](#) structure
- `swapchain` **must** be a pointer to an [XrEnvironmentDepthSwapchainMETA](#) handle



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The `XrEnvironmentDepthSwapchainCreateInfoMETA` structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthSwapchainCreateInfoMETA {
    XrStructureType          type;
    const void*              next;
    XrEnvironmentDepthSwapchainCreateFlagsMETA createFlags;
} XrEnvironmentDepthSwapchainCreateInfoMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `createFlags` is a bitmask of `XrEnvironmentDepthSwapchainCreateFlagBitsMETA`.

`XrEnvironmentDepthSwapchainCreateInfoMETA` contains creation options for the readable depth swapchain, and is passed to `xrCreateEnvironmentDepthSwapchainMETA`.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to using `XrEnvironmentDepthSwapchainCreateInfoMETA`
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_SWAPCHAIN_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `createFlags` **must** be `0`

The `XrEnvironmentDepthSwapchainCreateFlagsMETA` specifies creation options for `XrEnvironmentDepthSwapchainCreateInfoMETA`.

```
// Provided by XR_META_environment_depth
typedef XrFlags64 XrEnvironmentDepthSwapchainCreateFlagsMETA;
```

Valid bits for `XrEnvironmentDepthProviderCreateFlagsMETA` are defined by `XrEnvironmentDepthSwapchainCreateFlagBitsMETA`, which is specified as:

```
// Provided by XR_META_environment_depth
// Flag bits for XrEnvironmentDepthSwapchainCreateFlagsMETA
```

There are currently no flag bits defined. This is reserved for future use.

The `xrGetEnvironmentDepthSwapchainStateMETA` function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrGetEnvironmentDepthSwapchainStateMETA(
    XrEnvironmentDepthSwapchainMETA    swapchain,
    XrEnvironmentDepthSwapchainStateMETA* state);
```

## Parameter Descriptions

- `swapchain` is an `XrEnvironmentDepthSwapchainMETA` handle.
- `state` is a pointer to an `XrEnvironmentDepthSwapchainStateMETA`.

`xrGetEnvironmentDepthSwapchainStateMETA` retrieves information about the `XrEnvironmentDepthSwapchainMETA`. This information is constant throughout the lifetime of the `XrEnvironmentDepthSwapchainMETA`.

### Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling `xrGetEnvironmentDepthSwapchainStateMETA`
- `swapchain` **must** be a valid `XrEnvironmentDepthSwapchainMETA` handle
- `state` **must** be a pointer to an `XrEnvironmentDepthSwapchainStateMETA` structure

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `XrEnvironmentDepthSwapchainStateMETA` structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthSwapchainStateMETA {
    XrStructureType    type;
    void*              next;
    uint32_t           width;
    uint32_t           height;
} XrEnvironmentDepthSwapchainStateMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `width` is the width of the image.
- `height` is the height of the image.

## Valid Usage (Implicit)

- The [XR\\_META\\_environment\\_depth](#) extension **must** be enabled prior to using [XrEnvironmentDepthSwapchainStateMETA](#)
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_SWAPCHAIN_STATE_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [xrDestroyEnvironmentDepthSwapchainMETA](#) function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrDestroyEnvironmentDepthSwapchainMETA(
    XrEnvironmentDepthSwapchainMETA    swapchain);
```

## Parameter Descriptions

- `swapchain` is the [XrEnvironmentDepthSwapchainMETA](#) to be destroyed.

The [xrDestroyEnvironmentDepthSwapchainMETA](#) function destroys a readable environment depth swapchain.

All submitted graphics API commands that refer to `swapchain` **must** have completed execution. Runtimes **may** continue to utilize `swapchain` images after [xrDestroyEnvironmentDepthSwapchainMETA](#) is called.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling `xrDestroyEnvironmentDepthSwapchainMETA`
- `swapchain` **must** be a valid `XrEnvironmentDepthSwapchainMETA` handle

## Thread Safety

- Access to `swapchain`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## 12.91.7. Accessing the Readable Depth Swapchain During Rendering

The `xrEnumerateEnvironmentDepthSwapchainImagesMETA` function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrEnumerateEnvironmentDepthSwapchainImagesMETA(
    XrEnvironmentDepthSwapchainMETA    swapchain,
    uint32_t                             imageCapacityInput,
    uint32_t*                             imageCountOutput,
    XrSwapchainImageBaseHeader*         images);
```

## Parameter Descriptions

- `swapchain` is the [XrEnvironmentDepthSwapchainMETA](#) to get images from.
- `imageCapacityInput` is the capacity of the images array, or 0 to indicate a request to retrieve the required capacity.
- `imageCountOutput` is a pointer to the count of images written, or a pointer to the required capacity in the case that `imageCapacityInput` is insufficient.
- `images` is a pointer to an array of graphics API-specific [XrSwapchainImage](#) structures, all of the same type, based on [XrSwapchainImageBaseHeader](#). It **can** be `NULL` if `imageCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `images` size.

[xrEnumerateEnvironmentDepthSwapchainImagesMETA](#) fills an array of graphics API-specific [XrSwapchainImage\\*](#) structures derived from [XrSwapchainImageBaseHeader](#). The resources **must** be constant and valid for the lifetime of the [XrEnvironmentDepthSwapchainMETA](#). This function behaves analogously to [xrEnumerateSwapchainImages](#).

Runtimes **must** always return identical buffer contents from this enumeration for the lifetime of the swapchain.

Note: `images` is a pointer to an array of structures of graphics API-specific type, not an array of structure pointers.

The pointer submitted as `images` will be treated as an array of the expected graphics API-specific type based on the graphics API used at session creation time. If the type member of any array element accessed in this way does not match the expected value, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

## Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling [xrEnumerateEnvironmentDepthSwapchainImagesMETA](#)
- `swapchain` **must** be a valid [XrEnvironmentDepthSwapchainMETA](#) handle
- `imageCountOutput` **must** be a pointer to a `uint32_t` value
- If `imageCapacityInput` is not 0, `images` **must** be a pointer to an array of `imageCapacityInput` [XrSwapchainImageBaseHeader](#)-based structures. See also: [XrSwapchainImageD3D11KHR](#), [XrSwapchainImageD3D12KHR](#), [XrSwapchainImageOpenGLESKHR](#), [XrSwapchainImageOpenGLKHR](#), [XrSwapchainImageVulkanKHR](#)

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `xrAcquireEnvironmentDepthImageMETA` function is defined as:

```
// Provided by XR_META_environment_depth
XrResult xrAcquireEnvironmentDepthImageMETA(
    XrEnvironmentDepthProviderMETA    environmentDepthProvider,
    const XrEnvironmentDepthImageAcquireInfoMETA* acquireInfo,
    XrEnvironmentDepthImageMETA*     environmentDepthImage);
```

## Parameter Descriptions

- `environmentDepthProvider` is an `XrEnvironmentDepthProviderMETA` handle for the depth provider.
- `acquireInfo` is an `XrEnvironmentDepthImageAcquireInfoMETA` containing parameters for populating a depth swapchain image.
- `environmentDepthImage` is the returned `XrEnvironmentDepthImageMETA` containing information about the acquired depth image.

Acquires the latest available swapchain image that has been generated by the depth provider and ensures it is ready to be accessed by the application. The application **may** access and queue GPU operations using the acquired image until the next `xrEndFrame` call, when the image is released and the depth provider **may** write new depth data into it after completion of all work queued before the

[xrEndFrame](#) call.

The returned [XrEnvironmentDepthImageMETA](#) contains the swapchain index into the array enumerated by [xrEnumerateEnvironmentDepthSwapchainImagesMETA](#). It also contains other information such as the field of view and pose that are necessary to interpret the depth data.

There **must** be no more than one call to [xrAcquireEnvironmentDepthImageMETA](#) between any pair of corresponding [xrBeginFrame](#) and [xrEndFrame](#) calls in a session.

- The runtime **may** block if previously acquired swapchain images are still being used by the graphics API.
- The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if [xrAcquireEnvironmentDepthImageMETA](#) is called before [xrBeginFrame](#) or after [xrEndFrame](#).
- The runtime **must** return `XR_ERROR_CALL_ORDER_INVALID` if [xrAcquireEnvironmentDepthImageMETA](#) is called on a stopped [XrEnvironmentDepthProviderMETA](#).
- The runtime **must** return `XR_ERROR_LIMIT_REACHED` if [xrAcquireEnvironmentDepthImageMETA](#) is called more than once per frame - i.e. in a running session, after a call to [xrBeginFrame](#) that has not had an associated [xrEndFrame](#).
- Runtimes **must** return `XR_ENVIRONMENT_DEPTH_NOT_AVAILABLE_META` if no depth frame is available yet (i.e. the provider was recently started and did not yet have time to compute depth). Note that this is a success code. In this case the output parameters **must** be unchanged.
- The application **must** not utilize the swapchain image in calls to the graphics API after [xrEndFrame](#) has been called.
- A runtime **may** use the graphics API specific contexts provided to OpenXR. In particular:
  - For OpenGL, a runtime **may** use the OpenGL context specified in the call to [xrCreateSession](#), which needs external synchronization.
  - For Vulkan, a runtime **may** use the `VkQueue` specified in the [XrGraphicsBindingVulkan2KHR](#), which needs external synchronization.
  - For Direct3D12, a runtime **may** use the `ID3D12CommandQueue` specified in the [XrGraphicsBindingD3D12KHR](#), which needs external synchronization.

### Valid Usage (Implicit)

- The `XR_META_environment_depth` extension **must** be enabled prior to calling [xrAcquireEnvironmentDepthImageMETA](#)
- `environmentDepthProvider` **must** be a valid [XrEnvironmentDepthProviderMETA](#) handle
- `acquireInfo` **must** be a pointer to a valid [XrEnvironmentDepthImageAcquireInfoMETA](#) structure
- `environmentDepthImage` **must** be a pointer to an [XrEnvironmentDepthImageMETA](#) structure



## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING
- XR\_ENVIRONMENT\_DEPTH\_NOT\_AVAILABLE\_META

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_TIME\_INVALID
- XR\_ERROR\_CALL\_ORDER\_INVALID

The `XrEnvironmentDepthImageAcquireInfoMETA` structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthImageAcquireInfoMETA {
    XrStructureType    type;
    const void*        next;
    XrSpace             space;
    XrTime              displayTime;
} XrEnvironmentDepthImageAcquireInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `space` is an [XrSpace](#) defining the reference frame of the returned pose in [XrEnvironmentDepthImageMETA](#).
- `displayTime` is an [XrTime](#) specifying the time used to compute the pose for the returned pose in [XrEnvironmentDepthImageMETA](#). Clients **should** pass their predicted display time for the current frame.

## Valid Usage (Implicit)

- The [XR\\_META\\_environment\\_depth](#) extension **must** be enabled prior to using [XrEnvironmentDepthImageAcquireInfoMETA](#)
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_IMAGE_ACQUIRE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `space` **must** be a valid [XrSpace](#) handle

The [XrEnvironmentDepthImageViewMETA](#) structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthImageViewMETA {
    XrStructureType    type;
    const void*        next;
    XrFovf             fov;
    XrPosef            pose;
} XrEnvironmentDepthImageViewMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `fov` is an [XrFovf](#) specifying the field of view used to generate this view. The view is never flipped horizontally nor vertically.
- `pose` is an [XrPosef](#) specifying the pose from which the depth map was rendered. The reference frame is specified in [XrEnvironmentDepthImageAcquireInfoMETA](#).

## Valid Usage (Implicit)

- The [XR\\_META\\_environment\\_depth](#) extension **must** be enabled prior to using [XrEnvironmentDepthImageViewMETA](#)
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_IMAGE_VIEW_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEnvironmentDepthImageMETA](#) structure is defined as:

```
// Provided by XR_META_environment_depth
typedef struct XrEnvironmentDepthImageMETA {
    XrStructureType          type;
    const void*              next;
    uint32_t                 swapchainIndex;
    float                    nearZ;
    float                    farZ;
    XrEnvironmentDepthImageViewMETA views[2];
} XrEnvironmentDepthImageMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `swapchainIndex` is the index of the acquired texture in the depth swapchain.
- `nearZ` is the distance to the near Z plane in meters.
- `farZ` is the distance to the far Z plane in meters.
- `views` is an array of two [XrEnvironmentDepthImageViewMETA](#), one for each eye, where index 0 is left eye and index 1 is the right eye.

Depth is provided as textures in the same format as described in the [XR\\_KHR\\_composition\\_layer\\_depth](#) extension.

The frustum's Z-planes are placed at `nearZ` and `farZ` meters. When `farZ` is less than `nearZ`, an infinite projection matrix is used.

## Valid Usage (Implicit)

- The [XR\\_META\\_environment\\_depth](#) extension **must** be enabled prior to using [XrEnvironmentDepthImageViewMETA](#)
- `type` **must** be `XR_TYPE_ENVIRONMENT_DEPTH_IMAGE_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- Any given element of `views` **must** be a valid [XrEnvironmentDepthImageViewMETA](#) structure

### 12.91.8. Vulkan Swapchain Image Layout

For an application using Vulkan, after a successful call to [xrAcquireEnvironmentDepthImageViewMETA](#) that does **not** return `XR_ENVIRONMENT_DEPTH_NOT_AVAILABLE_META`, the following conditions apply to the **runtime**:

- The runtime **must** ensure the acquired readable depth swapchain image has a memory layout compatible with `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`. **Note** that this is different from [xrAcquireSwapchainImage](#) which guarantees `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`.
- The runtime **must** ensure the `VkQueue` specified in [XrGraphicsBindingVulkanKHR](#) / [XrGraphicsBindingVulkan2KHR](#) has ownership of the acquired readable depth swapchain image.

Upon next calling [xrEndFrame](#) after such an acquire call, the following conditions apply to the **application**:

- The application **must** ensure that the readable depth swapchain image has a memory layout compatible with `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`.
- The application **must** ensure that the readable depth swapchain image is owned by the `VkQueue` specified in `XrGraphicsBindingVulkanKHR` / `XrGraphicsBindingVulkan2KHR`.

The application is responsible for transitioning the swapchain image back to the image layout and queue ownership that the OpenXR runtime requires. If the image is not in a layout compatible with the above specifications, the runtime **may** exhibit undefined behavior.

### 12.91.9. Direct3D 12 Swapchain Image Resource State

For an application using D3D12, after a successful call to `xrAcquireEnvironmentDepthImageMETA` that does **not** return `XR_ENVIRONMENT_DEPTH_NOT_AVAILABLE_META`, the following conditions apply to the **runtime**:

- The runtime **must** ensure the acquired readable depth swapchain image has a resource state match with `D3D12_RESOURCE_STATE_ALL_SHADER_RESOURCE`. **Note** that this is different from `xrAcquireSwapchainImage` which guarantees `D3D12_RESOURCE_STATE_DEPTH_WRITE` for swapchain images with depth formats.
- The runtime **must** ensure that the `ID3D12CommandQueue` specified in `XrGraphicsBindingD3D12KHR` **may** read from the acquired readable depth swapchain image.

Upon next calling `xrEndFrame` after such an acquire call, the following conditions apply to the **application**:

- The application **must** ensure that the readable depth swapchain image has a resource state match with `D3D12_RESOURCE_STATE_ALL_SHADER_RESOURCE`.
- The application **must** ensure that the readable depth swapchain image is available for read/write on the `ID3D12CommandQueue` specified in `XrGraphicsBindingD3D12KHR`.

The application is responsible for transitioning the swapchain image back to the resource state and queue availability that the OpenXR runtime requires. If the image is not in a resource state match with the above specifications the runtime **may** exhibit undefined behavior.

#### Version History

- Revision 1, 2023-08-24 (Daniel Henell)
  - Initial extension description

## 12.92. XR\_META\_foveation\_eye\_tracked

### Name String

`XR_META_foveation_eye_tracked`

## Extension Type

Instance extension

## Registered Extension Number

201

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_FB\\_foveation](#)

and

[XR\\_FB\\_foveation\\_configuration](#)

## Contributors

Ross Ning, Facebook

Kevin Xiao, Facebook

Remi Palandri, Facebook

Jian Zhang, Facebook

Neel Bedekar, Facebook

## Overview

Eye tracked foveated rendering renders lower pixel density in the periphery of the user's gaze, taking advantage of low peripheral acuity.

This extension allows:

- An application to query eye tracked foveation availability.
- An application to request eye tracked foveation profile supported by the runtime and apply them to foveation-supported swapchains.
- An application to query foveation center position every frame.
- An application to request a foveation pattern update from the runtime. As a consequence, runtime knows how to adjust the eye tracking camera exposure start time in order to optimize the total pipeline latency.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Object Types

## New Flag Types

```
// Provided by XR_META_foveation_eye_tracked
typedef XrFlags64 XrFoveationEyeTrackedProfileCreateFlagsMETA;
```

```
// Provided by XR_META_foveation_eye_tracked
// Flag bits for XrFoveationEyeTrackedProfileCreateFlagsMETA
```

There are currently no eye tracked profile create flags. This is reserved for future use.

```
// Provided by XR_META_foveation_eye_tracked
typedef XrFlags64 XrFoveationEyeTrackedStateFlagsMETA;
```

```
// Provided by XR_META_foveation_eye_tracked
// Flag bits for XrFoveationEyeTrackedStateFlagsMETA
static const XrFoveationEyeTrackedStateFlagsMETA
XR_FOVEATION_EYE_TRACKED_STATE_VALID_BIT_META = 0x00000001;
```

## Flag Descriptions

- `XR_FOVEATION_EYE_TRACKED_STATE_VALID_BIT_META` — Indicates whether or not foveation data is valid. This can happen if the eye tracker is obscured, the camera has dirt, or eye lid is closed, etc.

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_FOVEATION_EYE_TRACKED_PROFILE_CREATE_INFO_META`
- `XR_TYPE_FOVEATION_EYE_TRACKED_STATE_META`
- `XR_TYPE_SYSTEM_FOVEATION_EYE_TRACKED_PROPERTIES_META`

## New Enums

## New Structures

The [XrFoveationEyeTrackedProfileCreateInfoMETA](#) structure is defined as:

```
// Provided by XR_META_foveation_eye_tracked
typedef struct XrFoveationEyeTrackedProfileCreateInfoMETA {
    XrStructureType                type;
    const void*                    next;
    XrFoveationEyeTrackedProfileCreateFlagsMETA flags;
} XrFoveationEyeTrackedProfileCreateInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of [XrFoveationEyeTrackedProfileCreateFlagBitsMETA](#) which indicate various characteristics for how eye tracked foveation is enabled on the swapchain.

[XrFoveationEyeTrackedProfileCreateInfoMETA](#) **can** be added to the `next` chain of [XrFoveationLevelProfileCreateInfoFB](#) in order to enable eye tracked foveation. The runtime **must** apply an eye tracked foveation pattern according to the parameters defined in the [XrFoveationLevelProfileCreateInfoFB](#).

## Valid Usage (Implicit)

- The [XR\\_META\\_foveation\\_eye\\_tracked](#) extension **must** be enabled prior to using [XrFoveationEyeTrackedProfileCreateInfoMETA](#)
- `type` **must** be `XR_TYPE_FOVEATION_EYE_TRACKED_PROFILE_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be `0`

The [XrFoveationEyeTrackedStateMETA](#) structure is defined as:



```
// Provided by XR_META_foveation_eye_tracked
typedef struct XrFoveationEyeTrackedStateMETA {
    XrStructureType          type;
    void*                    next;
    XrVector2f               foveationCenter[
XR_FOVEATION_CENTER_SIZE_META];
    XrFoveationEyeTrackedStateFlagsMETA flags;
} XrFoveationEyeTrackedStateMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `foveationCenter` is the center of the foveal region defined in NDC space in the range of -1 to 1 for both eyes.
- `flags` is a bitmask of [XrFoveationEyeTrackedStateFlagBitsMETA](#) which indicates various characteristics for current foveation state.

[XrFoveationEyeTrackedStateMETA](#) **must** be provided when calling [xrGetFoveationEyeTrackedStateMETA](#). The runtime **must** interpret [XrFoveationEyeTrackedStateMETA](#) without any additional structs in its `next` chain in order to query eye tracked foveation state, e.g. the center of the foveal region.

## Valid Usage (Implicit)

- The [XR\\_META\\_foveation\\_eye\\_tracked](#) extension **must** be enabled prior to using [XrFoveationEyeTrackedStateMETA](#)
- `type` **must** be `XR_TYPE_FOVEATION_EYE_TRACKED_STATE_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrSystemFoveationEyeTrackedPropertiesMETA](#) structure is defined as:

```
// Provided by XR_META_foveation_eye_tracked
typedef struct XrSystemFoveationEyeTrackedPropertiesMETA {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsFoveationEyeTracked;
} XrSystemFoveationEyeTrackedPropertiesMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsFoveationEyeTracked` indicates if the current system is capable of eye tracked foveation.

An application **can** inspect whether the system is capable of eye tracked foveation by extending the [XrSystemProperties](#) with [XrSystemFoveationEyeTrackedPropertiesMETA](#) structure when calling [xrGetSystemProperties](#).

## Valid Usage (Implicit)

- The [XR\\_META\\_foveation\\_eye\\_tracked](#) extension **must** be enabled prior to using [XrSystemFoveationEyeTrackedPropertiesMETA](#)
- `type` **must** be `XR_TYPE_SYSTEM_FOVEATION_EYE_TRACKED_PROPERTIES_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrGetFoveationEyeTrackedStateMETA](#) function is defined as:

```
// Provided by XR_META_foveation_eye_tracked
XrResult xrGetFoveationEyeTrackedStateMETA(
    XrSession                session,
    XrFoveationEyeTrackedStateMETA* foveationState);
```

## Parameter Descriptions

- `session` is the [XrSession](#) in which the eye tracked foveation profile is applied.
- `foveationState` is a pointer to an [XrFoveationEyeTrackedStateMETA](#) structure returning the current eye tracked foveation state.

The [xrGetFoveationEyeTrackedStateMETA](#) function returns the current eye tracked foveation state including the center of the foveal region, validity of the foveation data, etc.

Note that [xrUpdateSwapchainFB](#) **should** be called right before the [xrGetFoveationEyeTrackedStateMETA](#) function in order to (1) request a foveation pattern update by the runtime (2) optionally instruct the runtime to adjust the eye tracking camera capture start time in order to optimize for pipeline latency.

## Valid Usage (Implicit)

- The [XR\\_META\\_foveation\\_eye\\_tracked](#) extension **must** be enabled prior to calling [xrGetFoveationEyeTrackedStateMETA](#)
- `session` **must** be a valid [XrSession](#) handle
- `foveationState` **must** be a pointer to an [XrFoveationEyeTrackedStateMETA](#) structure

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

## Issues

## Version History

- Revision 1, 2022-04-08 (Ross Ning)
  - Initial extension description

# 12.93. XR\_META\_headset\_id

## Name String

XR\_META\_headset\_id

## Extension Type

Instance extension

## Registered Extension Number

246

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-08-11

## IP Status

No known IP claims.

## Contributors

Wenlin Mao, Meta Platforms  
Andreas Loeve Selvik, Meta Platforms  
Rémi Palandri, Meta Platforms  
John Kearney, Meta Platforms  
Jonathan Wright, Meta Platforms

## Contacts

Wenlin Mao, Meta Platforms

### Note



Using the headset ID to alter application behavior is discouraged, as it interferes with compatibility with current and future headsets. The OpenXR specification is designed with the goal of avoiding the need for explicit per-device logic. If the use of this extension is required, it is encouraged to let the OpenXR working group know about the use case, through a communication channel like email or GitHub. While this usage is discouraged, applications that need this functionality are encouraged to use this extension instead of the `systemName` field in `XrSystemProperties`. Game engines and similar middleware **should** not enable this extension by default. This extension will be deprecated and no longer exposed once the remaining use cases are resolved in a more portable way.

The `XrSystemHeadsetIdPropertiesMETA` structure is defined as:

```
// Provided by XR_META_headset_id
typedef struct XrSystemHeadsetIdPropertiesMETA {
    XrStructureType    type;
    void*              next;
    XrUuidEXT          id;
} XrSystemHeadsetIdPropertiesMETA;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `id` is the `XrUuidEXT` corresponding to the headset model.

An application **can** get a corresponding headset UUID of the headset model by chaining an `XrSystemHeadsetIdPropertiesMETA` structure to the `XrSystemProperties` when calling `xrGetSystemProperties`.

The UUID returned in the `XrSystemHeadsetIdPropertiesMETA` structure is an opaque UUID that identifies a runtime / headset model combo.

The runtime **should** always return the same UUID for a given headset model for the entire lifetime of that product.

The runtime **may** report a different UUID to some applications for compatibility purposes.

This is in contrast to the `XrSystemProperties::systemName` field which is not required to be consistent

across product renames.

This is intended to be a temporary feature that will be deprecated along with its extension as soon as motivating use cases are resolved in a better way. See the disclaimer at the start of the [XR\\_META\\_headset\\_id](#) extension documentation for more details.

### Valid Usage (Implicit)

- The [XR\\_META\\_headset\\_id](#) extension **must** be enabled prior to using [XrSystemHeadsetIdPropertiesMETA](#)
- **type** **must** be [XR\\_TYPE\\_SYSTEM\\_HEADSET\\_ID\\_PROPERTIES\\_META](#)
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Object Types

### New Atom

### New Flag Types

### New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SYSTEM\\_HEADSET\\_ID\\_PROPERTIES\\_META](#)

### New Enums

### New Structures

- [XrSystemHeadsetIdPropertiesMETA](#)

### New Functions

### Issues

### Version History

- Revision 1, 2022-08-11 (Wenlin Mao)
  - Initial extension description
- Revision 2, 2023-01-30 (Wenlin Mao)
  - Drop requirement for [XR\\_EXT\\_uuid](#) must be enabled

# 12.94. XR\_META\_local\_dimming

## Name String

`XR_META_local_dimming`

## Extension Type

Instance extension

## Registered Extension Number

217

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-05-05

## IP Status

No known IP claims.

## Contributors

Ross Ning, Meta Platforms

Haomiao Jiang, Meta Platforms

Remi Palandri, Meta Platforms

Xiang Wei, Meta Platforms

## Overview

Local dimming allows to adjust backlight intensity of dark areas on the screen in order to increase content dynamic range. Local dimming feature is not intended for optical see-through HMDs.

An application **can** request the local dimming mode on a frame basis by chaining an [XrLocalDimmingFrameEndInfoMETA](#) structure to the [XrFrameEndInfo](#).

- Using `XrFrameEndInfoLocalDimmingFB` is considered as a hint and will not trigger [xrEndFrame](#) errors whether or not the requested dimming mode is fulfilled by the runtime.
- The runtime will have full control of the local dimming mode and **may** disregard app requests. For example, the runtime **may** allow only one primary client to control the local dimming mode.

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_LOCAL_DIMMING_FRAME_END_INFO_META`

## New Enums

The local dimming mode is specified by the [XrLocalDimmingModeMETA](#) enumeration:

```
// Provided by XR_META_local_dimming
typedef enum XrLocalDimmingModeMETA {
    XR_LOCAL_DIMMING_MODE_OFF_META = 0,
    XR_LOCAL_DIMMING_MODE_ON_META = 1,
    XR_LOCAL_DIMMING_MODE_MAX_ENUM_META = 0x7FFFFFFF
} XrLocalDimmingModeMETA;
```

### Enumerant Descriptions

- `XR_LOCAL_DIMMING_MODE_OFF_META` — Local dimming is turned off by default for the current submitted frame. This is the same as not chaining [XrLocalDimmingModeMETA](#).
- `XR_LOCAL_DIMMING_MODE_ON_META` — Local dimming is turned on for the current submitted frame.

## New Structures

The [XrLocalDimmingFrameEndInfoMETA](#) structure is defined as:

```
// Provided by XR_META_local_dimming
typedef struct XrLocalDimmingFrameEndInfoMETA {
    XrStructureType      type;
    const void*          next;
    XrLocalDimmingModeMETA localDimmingMode;
} XrLocalDimmingFrameEndInfoMETA;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `localDimmingMode` is the local dimming mode for current submitted frame.

The [XrLocalDimmingFrameEndInfoMETA](#) is a structure that an application **can** chain in [XrFrameEndInfo](#) in order to request a local dimming mode.

## Valid Usage (Implicit)

- The `XR_META_local_dimming` extension **must** be enabled prior to using [XrLocalDimmingFrameEndInfoMETA](#)
- `type` **must** be `XR_TYPE_LOCAL_DIMMING_FRAME_END_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `localDimmingMode` **must** be a valid [XrLocalDimmingModeMETA](#) value

## New Functions

## Issues

## Version History

- Revision 1, 2022-05-05 (Ross Ning)
  - Initial draft

# 12.95. XR\_META\_passthrough\_color\_lut

## Name String

`XR_META_passthrough_color_lut`

## Extension Type

Instance extension

## Registered Extension Number

267

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_FB\\_passthrough](#)

## Last Modified Date

2022-11-28

## IP Status

No known IP claims.

## Contributors

Andreas Loeve Selvik, Meta Platforms  
Johannes Schmid, Meta Platforms  
John Kearney, Meta Platforms

## Overview

This extension adds the capability to define and apply RGB to RGB(A) color look-up tables (LUTs) to passthrough layers created using [XR\\_FB\\_passthrough](#).

Color LUTs are 3-dimensional arrays which map each input color to a different output color. When applied to a Passthrough layer, the runtime **must** transform Passthrough camera images according to this map before display. Color LUTs **may** be used to achieve effects such as color grading, level control, color filtering, or chroma keying.

Color LUTs **must** be created using [xrCreatePassthroughColorLutMETA](#) before they **can** be applied to a Passthrough layer in a call to [xrPassthroughLayerSetStyleFB](#) (as a part of [XrPassthroughColorMapLutMETA](#) or [XrPassthroughColorMapInterpolatedLutMETA](#)). A color LUT **may** be applied to multiple Passthrough layers simultaneously.

## New Object Types

```
XR_DEFINE_HANDLE(XrPassthroughColorLutMETA)
```

[XrPassthroughColorLutMETA](#) represents the definition and data for a color LUT which **may** be applied to a passthrough layer using [xrPassthroughLayerSetStyleFB](#).

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SYSTEM\\_PASSTHROUGH\\_COLOR\\_LUT\\_PROPERTIES\\_META](#)

- `XR_TYPE_PASSTHROUGH_COLOR_LUT_CREATE_INFO_META`
- `XR_TYPE_PASSTHROUGH_COLOR_LUT_UPDATE_INFO_META`
- `XR_TYPE_PASSTHROUGH_COLOR_MAP_LUT_META`
- `XR_TYPE_PASSTHROUGH_COLOR_MAP_INTERPOLATED_LUT_META`

## New Enums

Specify the color channels contained in the color LUT.

```
typedef enum XrPassthroughColorLutChannelsMETA {
    XR_PASSTHROUGH_COLOR_LUT_CHANNELS_RGB_META = 1,
    XR_PASSTHROUGH_COLOR_LUT_CHANNELS_RGBA_META = 2,
    XR_PASSTHROUGH_COLOR_LUT_CHANNELS_MAX_ENUM_META = 0x7FFFFFFF
} XrPassthroughColorLutChannelsMETA;
```

## New Structures

The `XrSystemPassthroughColorLutPropertiesMETA` structure is defined as:

```
// Provided by XR_META_passthrough_color_lut
typedef struct XrSystemPassthroughColorLutPropertiesMETA {
    XrStructureType    type;
    const void*        next;
    uint32_t           maxColorLutResolution;
} XrSystemPassthroughColorLutPropertiesMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `maxColorLutResolution` Maximum value for `XrPassthroughColorLutCreateInfoMETA::resolution` supported by the system. Runtimes implementing this extension **must** support a value of at least 32 for this property.

When the `XR_META_passthrough_color_lut` extension is enabled, an application **may** pass in an `XrSystemPassthroughColorLutPropertiesMETA` structure in next chain structure when calling `xrGetSystemProperties` to acquire information about the connected system.

The runtime **must** populate the [XrSystemPassthroughColorLutPropertiesMETA](#) structure with the relevant information to the [XrSystemProperties](#) returned by the [xrGetSystemProperties](#) call.

### Valid Usage (Implicit)

- The [XR\\_META\\_passthrough\\_color\\_lut](#) extension **must** be enabled prior to using [XrSystemPassthroughColorLutPropertiesMETA](#)
- **type** **must** be `XR_TYPE_SYSTEM_PASSTHROUGH_COLOR_LUT_PROPERTIES_META`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrPassthroughColorLutDataMETA](#) structure is defined as:

```
// Provided by XR_META_passthrough_color_lut
typedef struct XrPassthroughColorLutDataMETA {
    uint32_t      bufferSize;
    const uint8_t* buffer;
} XrPassthroughColorLutDataMETA;
```

### Member Descriptions

- **bufferSize** is the number of bytes contained in the buffer data.
- **buffer** is a pointer to a memory block of **bufferSize** bytes that contains the LUT data.

[XrPassthroughColorLutDataMETA](#) defines the LUT data for a color LUT. This structure is used when creating and updating color LUTs.

### Valid Usage (Implicit)

- The [XR\\_META\\_passthrough\\_color\\_lut](#) extension **must** be enabled prior to using [XrPassthroughColorLutDataMETA](#)
- **buffer** **must** be a pointer to an array of **bufferSize** `uint8_t` values
- The **bufferSize** parameter **must** be greater than `0`

The [XrPassthroughColorLutCreateInfoMETA](#) structure is defined as:

```
// Provided by XR_META_passthrough_color_lut
typedef struct XrPassthroughColorLutCreateInfoMETA {
    XrStructureType          type;
    const void*              next;
    XrPassthroughColorLutChannelsMETA channels;
    uint32_t                 resolution;
    XrPassthroughColorLutDataMETA data;
} XrPassthroughColorLutCreateInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `channels` defines the color channels expected in one LUT element. The number of bytes expected per LUT element is 3 for [XR\\_PASSTHROUGH\\_COLOR\\_LUT\\_CHANNELS\\_RGB\\_META](#) and 4 for [XR\\_PASSTHROUGH\\_COLOR\\_LUT\\_CHANNELS\\_RGBA\\_META](#).
- `resolution` is the number of LUT elements per input channel. The total number of elements in the LUT is  $\text{resolution}^3$ .
- `data` contains the data the LUT is initialized with.

`resolution` **must** be a power of 2, otherwise the runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#). The runtime **may** impose a limit on the maximum supported resolution, which is indicated in [XrSystemPassthroughColorLutPropertiesMETA](#). If `resolution` exceeds that limit, the runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#).

`data` contains a 3-dimensional array which defines an output color for each RGB input color. The input color is scaled to be in the range  $[0, \text{resolution}]$ . For an RGBA LUT, the RGBA tuple of output colors for an input color  $(R_{in}, G_{in}, B_{in})$  is found in the four bytes starting at the offset  $4 * (R_{in} + G_{in} * \text{resolution} + B_{in} * \text{resolution}^2)$ . For an RGB LUT, the RGB tuple of output colors for an input color  $(R_{in}, G_{in}, B_{in})$  is found in the three bytes starting at the offset  $3 * (R_{in} + G_{in} * \text{resolution} + B_{in} * \text{resolution}^2)$ .

Color LUT data **must** be specified and interpreted in sRGB color space.

Runtimes **must** employ trilinear interpolation of neighboring color values if the resolution of the color LUT is smaller than the bit depth of the input colors.

The value of [XrPassthroughColorLutDataMETA::bufferSize](#) in `data` **must** be equal to  $\text{resolution}^3 * \text{bytesPerElement}$ , where `bytesPerElement` is either 3 or 4 depending on `channels`. Otherwise, the runtime **must** return [XR\\_ERROR\\_PASSTHROUGH\\_COLOR\\_LUT\\_BUFFER\\_SIZE\\_MISMATCH\\_META](#).

## Valid Usage (Implicit)

- The `XR_META_passthrough_color_lut` extension **must** be enabled prior to using `XrPassthroughColorLutCreateInfoMETA`
- `type` **must** be `XR_TYPE_PASSTHROUGH_COLOR_LUT_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `channels` **must** be a valid `XrPassthroughColorLutChannelsMETA` value
- `data` **must** be a valid `XrPassthroughColorLutDataMETA` structure

The `XrPassthroughColorLutUpdateInfoMETA` structure is defined as:

```
// Provided by XR_META_passthrough_color_lut
typedef struct XrPassthroughColorLutUpdateInfoMETA {
    XrStructureType          type;
    const void*              next;
    XrPassthroughColorLutDataMETA data;
} XrPassthroughColorLutUpdateInfoMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `data` contains the updated LUT data.

The LUT data **may** be updated for an existing color LUT, while channels and resolution remain constant after creation. Hence, the value of `XrPassthroughColorLutDataMETA::bufferSize` in `data` **must** be equal to the buffer size specified at creation. Otherwise, the runtime **must** return `XR_ERROR_PASSTHROUGH_COLOR_LUT_BUFFER_SIZE_MISMATCH_META`.

## Valid Usage (Implicit)

- The `XR_META_passthrough_color_lut` extension **must** be enabled prior to using `XrPassthroughColorLutUpdateInfoMETA`
- `type` **must** be `XR_TYPE_PASSTHROUGH_COLOR_LUT_UPDATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `data` **must** be a valid `XrPassthroughColorLutDataMETA` structure

The `XrPassthroughColorMapLutMETA` structure is defined as:

```
// Provided by XR_META_passthrough_color_lut
typedef struct XrPassthroughColorMapLutMETA {
    XrStructureType      type;
    const void*         next;
    XrPassthroughColorLutMETA colorLut;
    float               weight;
} XrPassthroughColorMapLutMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `colorLut` is an `XrPassthroughColorLutMETA`.
- `weight` is a factor in the range [0, 1] which defines the linear blend between the original and the mapped colors for the output color.

`XrPassthroughColorMapLutMETA` lets applications apply a color LUT to a passthrough layer. Other Passthrough style elements (such as edges) **must** not be affected by color LUTs.

Applications **may** use `weight` to efficiently blend between the original colors and the mapped colors. The blend is computed as  $(1 - \text{weight}) * C_{in} + \text{weight} * \text{colorLut}[C_{in}]$ .

`XrPassthroughColorMapLutMETA` is provided in the `next` chain of `XrPassthroughStyleFB` when calling `xrPassthroughLayerSetStyleFB`. Subsequent calls to `xrPassthroughLayerSetStyleFB` with `XrPassthroughColorMapLutMETA` in the `next` chain update the color LUT for that layer. Subsequent calls to `xrPassthroughLayerSetStyleFB` without this `XrPassthroughColorMapLutMETA` (or `XrPassthroughColorMapInterpolatedLutMETA`) in the next chain disable color LUTs for that layer.

## Valid Usage (Implicit)

- The `XR_META_passthrough_color_lut` extension **must** be enabled prior to using `XrPassthroughColorMapLutMETA`
- `type` **must** be `XR_TYPE_PASSTHROUGH_COLOR_MAP_LUT_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `colorLut` **must** be a valid `XrPassthroughColorLutMETA` handle

The `XrPassthroughColorMapInterpolatedLutMETA` structure is defined as:

```
// Provided by XR_META_passthrough_color_lut
typedef struct XrPassthroughColorMapInterpolatedLutMETA {
    XrStructureType      type;
    const void*         next;
    XrPassthroughColorLutMETA sourceColorLut;
    XrPassthroughColorLutMETA targetColorLut;
    float               weight;
} XrPassthroughColorMapInterpolatedLutMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `sourceColorLut` is the initial `XrPassthroughColorLutMETA`.
- `targetColorLut` is the final `XrPassthroughColorLutMETA`.
- `weight` is a factor in the range [0, 1] which defines the linear blend between the initial and the final color LUT.

`XrPassthroughColorMapInterpolatedLutMETA` lets applications apply the interpolation between two color LUTs to a passthrough layer. Applications **may** use this feature to smoothly transition between two color LUTs. Other Passthrough style elements (such as edges) **must** not be affected by color LUTs.

The blend between `sourceColorLut` and `targetColorLut` is computed as  $(1 - \text{weight}) * \text{sourceColorLut} [C_{in}] + \text{weight} * \text{targetColorLut} [C_{in}]$ .

`XrPassthroughColorMapInterpolatedLutMETA` is provided in the `next` chain of `XrPassthroughStyleFB` when calling `xrPassthroughLayerSetStyleFB`. Subsequent calls to `xrPassthroughLayerSetStyleFB` with



[XrPassthroughColorMapInterpolatedLutMETA](#) in the next chain update the color LUT for that layer. Subsequent calls to [xrPassthroughLayerSetStyleFB](#) without this [XrPassthroughColorMapInterpolatedLutMETA](#) (or [XrPassthroughColorMapLutMETA](#)) in the next chain disable color LUTs for that layer.

### Valid Usage (Implicit)

- The [XR\\_META\\_passthrough\\_color\\_lut](#) extension **must** be enabled prior to using [XrPassthroughColorMapInterpolatedLutMETA](#)
- `type` **must** be [XR\\_TYPE\\_PASSTHROUGH\\_COLOR\\_MAP\\_INTERPOLATED\\_LUT\\_META](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `sourceColorLut` **must** be a valid [XrPassthroughColorLutMETA](#) handle
- `targetColorLut` **must** be a valid [XrPassthroughColorLutMETA](#) handle
- Both of `sourceColorLut` and `targetColorLut` **must** have been created, allocated, or retrieved from the same [XrPassthroughFB](#)

### New Functions

The [xrCreatePassthroughColorLutMETA](#) function is defined as:

```
// Provided by XR_META_passthrough_color_lut
XrResult xrCreatePassthroughColorLutMETA(
    XrPassthroughFB                passthrough,
    const XrPassthroughColorLutCreateInfoMETA* createInfo,
    XrPassthroughColorLutMETA*      colorLut);
```

### Parameter Descriptions

- `passthrough` is the [XrPassthroughFB](#) this color LUT is created for.
- `createInfo` is the [XrPassthroughColorLutCreateInfoMETA](#).
- `colorLut` is the resulting [XrPassthroughColorLutMETA](#).

Creates a passthrough color LUT. The resulting [XrPassthroughColorLutMETA](#) **may** be referenced in [XrPassthroughColorMapLutMETA](#) and [XrPassthroughColorMapInterpolatedLutMETA](#) in subsequent calls to [xrPassthroughLayerSetStyleFB](#).

## Valid Usage (Implicit)

- The `XR_META_passthrough_color_lut` extension **must** be enabled prior to calling `xrCreatePassthroughColorLutMETA`
- `passthrough` **must** be a valid `XrPassthroughFB` handle
- `createInfo` **must** be a pointer to a valid `XrPassthroughColorLutCreateInfoMETA` structure
- `colorLut` **must** be a pointer to an `XrPassthroughColorLutMETA` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_PASSTHROUGH_COLOR_LUT_BUFFER_SIZE_MISMATCH_META`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrDestroyPassthroughColorLutMETA` function is defined as:

```
// Provided by XR_META_passthrough_color_lut
XrResult xrDestroyPassthroughColorLutMETA(
    XrPassthroughColorLutMETA          colorLut);
```

## Parameter Descriptions

- `colorLut` is the [XrPassthroughColorLutMETA](#) to be destroyed.

Destroys a passthrough color LUT. If the color LUT is still in use (i.e. if for at least one passthrough layer, [xrPassthroughLayerSetStyleFB](#) has last been called with an instance of [XrPassthroughColorMapLutMETA](#) or [XrPassthroughColorMapInterpolatedLutMETA](#) in the next chain that references this color LUT), the runtime **must** retain the color LUT data and continue applying it to the affected passthrough layer until a different style is applied.

## Valid Usage (Implicit)

- The [XR\\_META\\_passthrough\\_color\\_lut](#) extension **must** be enabled prior to calling [xrDestroyPassthroughColorLutMETA](#)
- `colorLut` **must** be a valid [XrPassthroughColorLutMETA](#) handle

## Thread Safety

- Access to `colorLut`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The [xrUpdatePassthroughColorLutMETA](#) function is defined as:

```
// Provided by XR_META_passthrough_color_lut
XrResult xrUpdatePassthroughColorLutMETA(
    XrPassthroughColorLutMETA          colorLut,
    const XrPassthroughColorLutUpdateInfoMETA* updateInfo);
```

## Parameter Descriptions

- `colorLut` is the `XrPassthroughColorLutMETA` to be updated.
- `updateInfo` is the `XrPassthroughColorLutUpdateInfoMETA`.

Updates the LUT data of a passthrough color LUT. The data type of the color LUT (resolution and channels) is immutable. The provided data in this call **must** therefore match the data type specified at creation time. Specifically, `XrPassthroughColorLutDataMETA::bufferSize` of the new data **must** be equal to the `XrPassthroughColorLutDataMETA::bufferSize` specified during creation. Otherwise, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

The runtime **must** reflect changes to color LUT data on all Passthrough layers the color LUT is currently applied to.

## Valid Usage (Implicit)

- The `XR_META_passthrough_color_lut` extension **must** be enabled prior to calling `xrUpdatePassthroughColorLutMETA`
- `colorLut` **must** be a valid `XrPassthroughColorLutMETA` handle
- `updateInfo` **must** be a pointer to a valid `XrPassthroughColorLutUpdateInfoMETA` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PASSTHROUGH_COLOR_LUT_BUFFER_SIZE_MISMATCH_META`
- `XR_ERROR_FEATURE_UNSUPPORTED`

### Version History

- Revision 1, 2022-12-08 (Johannes Schmid)
  - Initial extension description

## 12.96. XR\_META\_passthrough\_preferences

### Name String

`XR_META_passthrough_preferences`

### Extension Type

Instance extension

### Registered Extension Number

218

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-04-25

## IP Status

No known IP claims.

## Contributors

Johannes Schmid, Meta Platforms

## Overview

This extension provides applications with access to system preferences concerning passthrough. For more information on how applications can control the display of passthrough, see [XR\\_FB\\_passthrough](#).

## New Flag Types

```
// Provided by XR_META_passthrough_preferences
typedef XrFlags64 XrPassthroughPreferenceFlagsMETA;
```

```
// Provided by XR_META_passthrough_preferences
// Flag bits for XrPassthroughPreferenceFlagsMETA
static const XrPassthroughPreferenceFlagsMETA
XR_PASSTHROUGH_PREFERENCE_DEFAULT_TO_ACTIVE_BIT_META = 0x00000001;
```

## Flag Descriptions

- `XR_PASSTHROUGH_PREFERENCE_DEFAULT_TO_ACTIVE_BIT_META` — Indicates that the runtime recommends apps to default to a mixed reality experience with passthrough (if supported).

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_PASSTHROUGH_PREFERENCES_META`

## New Structures

The [XrPassthroughPreferencesMETA](#) structure is defined as:

```
// Provided by XR_META_passthrough_preferences
typedef struct XrPassthroughPreferencesMETA {
    XrStructureType          type;
    const void*              next;
    XrPassthroughPreferenceFlagsMETA  flags;
} XrPassthroughPreferencesMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `flags` is a bitmask of [XrPassthroughPreferenceFlagBitsMETA](#) describing boolean passthrough preferences.

The runtime **must** populate the [XrPassthroughPreferencesMETA](#) structure with the relevant information when the app calls [xrGetPassthroughPreferencesMETA](#).

Presence of the bit flag [XR\\_PASSTHROUGH\\_PREFERENCE\\_DEFAULT\\_TO\\_ACTIVE\\_BIT\\_META](#) does not indicate a guarantee that applications **can** enable and use passthrough in practice. The runtime **may** impose restrictions on passthrough usage (e.g. based on hardware availability or permission models) independently of the state of this flag bit. Apps **should** test for this flag explicitly, as more flag bits **may** be introduced in the future.

## Valid Usage (Implicit)

- The [XR\\_META\\_passthrough\\_preferences](#) extension **must** be enabled prior to using [XrPassthroughPreferencesMETA](#)
- `type` **must** be [XR\\_TYPE\\_PASSTHROUGH\\_PREFERENCES\\_META](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrGetPassthroughPreferencesMETA](#) function is defined as:

```
// Provided by XR_META_passthrough_preferences
XrResult xrGetPassthroughPreferencesMETA(
    XrSession session,
    XrPassthroughPreferencesMETA* preferences);
```

## Parameter Descriptions

- `session` is the [XrSession](#).
- `preferences` points to an instance of [XrPassthroughPreferencesMETA](#) structure, that will be filled with returned information

An application **can** call [xrGetPassthroughPreferencesMETA](#) to retrieve passthrough-related preferences from the system.

## Valid Usage (Implicit)

- The [XR\\_META\\_passthrough\\_preferences](#) extension **must** be enabled prior to calling [xrGetPassthroughPreferencesMETA](#)
- `session` **must** be a valid [XrSession](#) handle
- `preferences` **must** be a pointer to an [XrPassthroughPreferencesMETA](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

## Version History



- Revision 1, 2023-04-25 (Johannes Schmid)
  - Initial extension description

## 12.97. XR\_META\_performance\_metrics

### Name String

`XR_META_performance_metrics`

### Extension Type

Instance extension

### Registered Extension Number

233

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Xiang Wei, Meta Platforms

### Overview

This extension provides APIs to enumerate and query performance metrics counters of the current XR device and XR application. Developers **can** perform performance analysis and do targeted optimization to the XR application using the performance metrics counters being collected. The application **should** not change its behavior based on the counter reads.

The performance metrics counters are organized into predefined [XrPath](#) values, under the root path `/perfmetrics_meta`. An application **can** query the available counters through [xrEnumeratePerformanceMetricsCounterPathsMETA](#). Here is a list of the performance metrics counter paths that **may** be provided on Meta devices:

- `/perfmetrics_meta/app/cpu_frametime`
- `/perfmetrics_meta/app/gpu_frametime`
- `/perfmetrics_meta/app/motion_to_photon_latency`
- `/perfmetrics_meta/compositor/cpu_frametime`
- `/perfmetrics_meta/compositor/gpu_frametime`
- `/perfmetrics_meta/compositor/dropped_frame_count`
- `/perfmetrics_meta/compositor/spacewarp_mode`

- */perfmetrics\_meta/device/cpu\_utilization\_average*
- */perfmetrics\_meta/device/cpu\_utilization\_worst*
- */perfmetrics\_meta/device/gpu\_utilization*
- */perfmetrics\_meta/device/cpu0\_utilization* through */perfmetrics\_meta/device/cpuX\_utilization*

After a session is created, an application **can** use [xrSetPerformanceMetricsStateMETA](#) to enable the performance metrics system for that session. An application **can** use [xrQueryPerformanceMetricsCounterMETA](#) to query a performance metrics counter on a session that has the performance metrics system enabled, or use [xrGetPerformanceMetricsStateMETA](#) to query if the performance metrics system is enabled.

Note: the measurement intervals of individual performance metrics counters are defined by the OpenXR runtime. The application **must** not make assumptions or change its behavior at runtime by measuring them.

In order to enable the functionality of this extension, the application **must** pass the name of the extension into [xrCreateInstance](#) via the [XrInstanceCreateInfo::enabledExtensionNames](#) parameter as indicated in the [Extensions](#) section.

## New Flag Types

```
typedef XrFlags64 XrPerformanceMetricsCounterFlagsMETA;
```

```
// Flag bits for XrPerformanceMetricsCounterFlagsMETA
static const XrPerformanceMetricsCounterFlagsMETA
XR_PERFORMANCE_METRICS_COUNTER_ANY_VALUE_VALID_BIT_META = 0x00000001;
static const XrPerformanceMetricsCounterFlagsMETA
XR_PERFORMANCE_METRICS_COUNTER_UINT_VALUE_VALID_BIT_META = 0x00000002;
static const XrPerformanceMetricsCounterFlagsMETA
XR_PERFORMANCE_METRICS_COUNTER_FLOAT_VALUE_VALID_BIT_META = 0x00000004;
```

## Flag Descriptions

- `XR_PERFORMANCE_METRICS_COUNTER_ANY_VALUE_VALID_BIT_META` — Indicates any of the values in `XrPerformanceMetricsCounterMETA` is valid.
- `XR_PERFORMANCE_METRICS_COUNTER_UINT_VALUE_VALID_BIT_META` — Indicates the `uintValue` in `XrPerformanceMetricsCounterMETA` is valid.
- `XR_PERFORMANCE_METRICS_COUNTER_FLOAT_VALUE_VALID_BIT_META` — Indicates the `floatValue` in `XrPerformanceMetricsCounterMETA` is valid.

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_PERFORMANCE_METRICS_STATE_META`
- `XR_TYPE_PERFORMANCE_METRICS_COUNTER_META`

## New Enums

```
// Provided by XR_META_performance_metrics
typedef enum XrPerformanceMetricsCounterUnitMETA {
    XR_PERFORMANCE_METRICS_COUNTER_UNIT_GENERIC_META = 0,
    XR_PERFORMANCE_METRICS_COUNTER_UNIT_PERCENTAGE_META = 1,
    XR_PERFORMANCE_METRICS_COUNTER_UNIT_MILLISECONDS_META = 2,
    XR_PERFORMANCE_METRICS_COUNTER_UNIT_BYTES_META = 3,
    XR_PERFORMANCE_METRICS_COUNTER_UNIT_HERTZ_META = 4,
    XR_PERFORMANCE_METRICS_COUNTER_UNIT_MAX_ENUM_META = 0x7FFFFFFF
} XrPerformanceMetricsCounterUnitMETA;
```

| Enum   | Description  |
|--|--|
| <code>XR_PERFORMANCE_METRICS_COUNTER_UNIT_GENERIC_META</code>      | the performance counter unit is generic (unspecified). |
| <code>XR_PERFORMANCE_METRICS_COUNTER_UNIT_PERCENTAGE_META</code>   | the performance counter unit is percentage (%).        |
| <code>XR_PERFORMANCE_METRICS_COUNTER_UNIT_MILLISECONDS_META</code> | the performance counter unit is millisecond.           |
| <code>XR_PERFORMANCE_METRICS_COUNTER_UNIT_BYTES_META</code>        | the performance counter unit is byte.                  |
| <code>XR_PERFORMANCE_METRICS_COUNTER_UNIT_HERTZ_META</code>        | the performance counter unit is hertz (Hz).            |

## New Structures

The [XrPerformanceMetricsStateMETA](#) structure is defined as:

```
// Provided by XR_META_performance_metrics
typedef struct XrPerformanceMetricsStateMETA {
    XrStructureType    type;
    const void*        next;
    XrBool32           enabled;
} XrPerformanceMetricsStateMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `enabled` is set to `XR_TRUE` to indicate the performance metrics system is enabled, `XR_FALSE` otherwise, when getting state. When setting state, set to `XR_TRUE` to enable the performance metrics system and `XR_FALSE` to disable it.

[XrPerformanceMetricsStateMETA](#) is provided as input when calling [xrSetPerformanceMetricsStateMETA](#) to enable or disable the performance metrics system. [XrPerformanceMetricsStateMETA](#) is populated as an output parameter when calling [xrGetPerformanceMetricsStateMETA](#) to query if the performance metrics system is enabled.

## Valid Usage (Implicit)

- The [XR\\_META\\_performance\\_metrics](#) extension **must** be enabled prior to using [XrPerformanceMetricsStateMETA](#)
- `type` **must** be `XR_TYPE_PERFORMANCE_METRICS_STATE_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrPerformanceMetricsCounterMETA](#) structure is defined as:

```
// Provided by XR_META_performance_metrics
typedef struct XrPerformanceMetricsCounterMETA {
    XrStructureType                type;
    const void*                    next;
    XrPerformanceMetricsCounterFlagsMETA counterFlags;
    XrPerformanceMetricsCounterUnitMETA counterUnit;
    uint32_t                       uintValue;
    float                           floatValue;
} XrPerformanceMetricsCounterMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `counterFlags` is a bitmask of [XrPerformanceMetricsCounterFlagBitsMETA](#) describing the validity of value members.
- `counterUnit` is an enum of [XrPerformanceMetricsCounterUnitMETA](#) describing the measurement unit.
- `uintValue` is the counter value in `uint32_t` format. It is valid if `counterFlags` contains `XR_PERFORMANCE_METRICS_COUNTER_UINT_VALUE_VALID_BIT_META`.
- `floatValue` is the counter value in `float` format. It is valid if `counterFlags` contains `XR_PERFORMANCE_METRICS_COUNTER_FLOAT_VALUE_VALID_BIT_META`.

[XrPerformanceMetricsCounterMETA](#) is populated by calling [xrQueryPerformanceMetricsCounterMETA](#) to query real-time performance metrics counter information.

## Valid Usage (Implicit)

- The [XR\\_META\\_performance\\_metrics](#) extension **must** be enabled prior to using [XrPerformanceMetricsCounterMETA](#)
- `type` **must** be `XR_TYPE_PERFORMANCE_METRICS_COUNTER_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `counterFlags` **must** be `0` or a valid combination of [XrPerformanceMetricsCounterFlagBitsMETA](#) values
- `counterUnit` **must** be a valid [XrPerformanceMetricsCounterUnitMETA](#) value

## New Functions

The [xrEnumeratePerformanceMetricsCounterPathsMETA](#) function enumerates all performance metrics counter paths that supported by the runtime, it is defined as:

```
// Provided by XR_META_performance_metrics
XrResult xrEnumeratePerformanceMetricsCounterPathsMETA(
    XrInstance          instance,
    uint32_t            counterPathCapacityInput,
    uint32_t*           counterPathCountOutput,
    XrPath*             counterPaths);
```

### Parameter Descriptions

- **instance** is an [XrInstance](#) handle previously created with [xrCreateInstance](#).
- **counterPathCapacityInput** is the capacity of the **counterPaths** array, or 0 to indicate a request to retrieve the required capacity.
- **counterPathCountOutput** is filled in by the runtime with the count of **counterPaths** written or the required capacity in the case that **counterPathCapacityInput** is insufficient.
- **counterPaths** is an array of [XrPath](#) filled in by the runtime which contains all the available performance metrics counters, but **can** be **NULL** if **counterPathCapacityInput** is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required **counterPaths** size.

### Valid Usage (Implicit)

- The [XR\\_META\\_performance\\_metrics](#) extension **must** be enabled prior to calling [xrEnumeratePerformanceMetricsCounterPathsMETA](#)
- **instance** **must** be a valid [XrInstance](#) handle
- **counterPathCountOutput** **must** be a pointer to a [uint32\\_t](#) value
- If **counterPathCapacityInput** is not 0, **counterPaths** **must** be a pointer to an array of **counterPathCapacityInput** [XrPath](#) values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

The `xrSetPerformanceMetricsStateMETA` function is defined as:

```
// Provided by XR_META_performance_metrics
XrResult xrSetPerformanceMetricsStateMETA(
    XrSession session,
    const XrPerformanceMetricsStateMETA* state);
```

## Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `state` is a pointer to an `XrPerformanceMetricsStateMETA` structure.

The `xrSetPerformanceMetricsStateMETA` function enables or disables the performance metrics system.

## Valid Usage (Implicit)

- The `XR_META_performance_metrics` extension **must** be enabled prior to calling `xrSetPerformanceMetricsStateMETA`
- `session` **must** be a valid `XrSession` handle
- `state` **must** be a pointer to a valid `XrPerformanceMetricsStateMETA` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrGetPerformanceMetricsStateMETA` function is defined as:

```
// Provided by XR_META_performance_metrics
XrResult xrGetPerformanceMetricsStateMETA(
    XrSession session,
    XrPerformanceMetricsStateMETA* state);
```

## Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `state` is a pointer to an `XrPerformanceMetricsStateMETA` structure.

The `xrGetPerformanceMetricsStateMETA` function gets the current state of the performance metrics system.

## Valid Usage (Implicit)

- The `XR_META_performance_metrics` extension **must** be enabled prior to calling `xrGetPerformanceMetricsStateMETA`
- `session` **must** be a valid `XrSession` handle
- `state` **must** be a pointer to an `XrPerformanceMetricsStateMETA` structure



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `xrQueryPerformanceMetricsCounterMETA` function is defined as:

```
// Provided by XR_META_performance_metrics
XrResult xrQueryPerformanceMetricsCounterMETA(
    XrSession          session,
    XrPath             counterPath,
    XrPerformanceMetricsCounterMETA* counter);
```

## Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `counterPath` is a valid performance metrics counter path.
- `counter` is a pointer to an `XrPerformanceMetricsCounterMETA` structure.

The `xrQueryPerformanceMetricsCounterMETA` function queries a performance metrics counter.

The application **should** enable the performance metrics system (by calling `xrSetPerformanceMetricsStateMETA`) before querying metrics using `xrQueryPerformanceMetricsCounterMETA`. If the performance metrics system has not been enabled before calling `xrQueryPerformanceMetricsCounterMETA`, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

If `counterPath` is not in the list returned by `xrEnumeratePerformanceMetricsCounterPathsMETA`, the runtime must return `XR_ERROR_PATH_UNSUPPORTED`.

## Valid Usage (Implicit)

- The `XR_META_performance_metrics` extension **must** be enabled prior to calling `xrQueryPerformanceMetricsCounterMETA`
- `session` **must** be a valid `XrSession` handle
- `counter` **must** be a pointer to an `XrPerformanceMetricsCounterMETA` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`

## Issues

## Version History

- Revision 1, 2022-04-28 (Xiang Wei)
  - Initial extension description
- Revision 2, 2022-09-16 (John Kearney)
  - Clarification of error codes

# 12.98. XR\_META\_recommended\_layer\_resolution

## Name String

`XR_META_recommended_layer_resolution`

## Extension Type

Instance extension

## Registered Extension Number

255

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Rohit Rao Padebettu, Meta

Remi Palandri, Meta

Ben Cumings, Meta

## Overview

The extension allows an application to request a recommended swapchain resolution from the runtime, in order to either allocate a swapchain of a more appropriate size, or to render into a smaller image rect according to the recommendation. For layers with multiple views such as [XrCompositionLayerProjection](#), the application **may** scale the individual views to match the scaled swapchain resolution.

The runtime **may** use any factors to drive the recommendation it wishes to return to the application. Those include static properties such as screen resolution and HMD type, but also dynamic ones such as layer positioning and system-wide GPU utilization.

Application **may** also use this extension to allocate the swapchain by passing in a layer with a swapchain handle [XR\\_NULL\\_HANDLE](#).

## New Structures

The [XrRecommendedLayerResolutionMETA](#) structure is defined as:

```
// Provided by XR_META_recommended_layer_resolution
typedef struct XrRecommendedLayerResolutionMETA {
    XrStructureType    type;
    void*              next;
    XrExtent2Di        recommendedImageDimensions;
    XrBool32           isValid;
} XrRecommendedLayerResolutionMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `recommendedImageDimensions` is the [XrExtent2Di](#) recommended image dimensions of the layer.
- `isValid` is the [XrBool32](#) boolean returned by the runtime which indicates whether the runtime returned a valid recommendation or does not have any recommendations to make.

If the runtime does not wish to make a recommendation, `isValid` **must** be `XR_FALSE` and `recommendedImageDimensions` **must** be {0,0}.

## Valid Usage (Implicit)

- The [XR\\_META\\_recommended\\_layer\\_resolution](#) extension **must** be enabled prior to using [XrRecommendedLayerResolutionMETA](#)
- `type` **must** be `XR_TYPE_RECOMMENDED_LAYER_RESOLUTION_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrRecommendedLayerResolutionGetInfoMETA](#) structure is defined as:

```
// Provided by XR_META_recommended_layer_resolution
typedef struct XrRecommendedLayerResolutionGetInfoMETA {
    XrStructureType          type;
    const void*              next;
    const XrCompositionLayerBaseHeader* layer;
    XrTime                   predictedDisplayTime;
} XrRecommendedLayerResolutionGetInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `layer` is a pointer to a structure based on [XrCompositionLayerBaseHeader](#), describing the layer for which the application wants a runtime-recommended swapchain resolution. Layers with multiple views **may** scale the views to match the scaled swapchain resolution.
- `predictedDisplayTime` is the [XrTime](#) that the application intends to submit the layer for.

If `predictedDisplayTime` is older than the predicted display time returned from most recent [xrWaitFrame](#) then, the runtime **must** return `XR_ERROR_TIME_INVALID`.

## Valid Usage (Implicit)

- The [XR\\_META\\_recommended\\_layer\\_resolution](#) extension **must** be enabled prior to using [XrRecommendedLayerResolutionGetInfoMETA](#)
- `type` **must** be `XR_TYPE_RECOMMENDED_LAYER_RESOLUTION_GET_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `layer` **must** be a pointer to a valid [XrCompositionLayerBaseHeader](#)-based structure. See also:  
[XrCompositionLayerCubeKHR](#), [XrCompositionLayerCylinderKHR](#),  
[XrCompositionLayerEquirect2KHR](#), [XrCompositionLayerEquirectKHR](#),  
[XrCompositionLayerPassthroughHTC](#), [XrCompositionLayerProjection](#),  
[XrCompositionLayerQuad](#)

## New Functions

The [xrGetRecommendedLayerResolutionMETA](#) function is defined as:

```
// Provided by XR_META_recommended_layer_resolution
XrResult xrGetRecommendedLayerResolutionMETA(
    XrSession session,
    const XrRecommendedLayerResolutionGetInfoMETA* info,
    XrRecommendedLayerResolutionMETA* resolution);
```

## Parameter Descriptions

- **session** is the [XrSession](#) in which the recommendation is made.
- **info** is a pointer to an [XrRecommendedLayerResolutionGetInfoMETA](#) structure containing the details of the layer for which the application is requesting a recommendation.
- **resolution** is a pointer to an [XrRecommendedLayerResolutionMETA](#) that the runtime will populate.

The [xrGetRecommendedLayerResolutionMETA](#) function returns the recommendation that the runtime wishes to make to the application for the layer provided in the [XrRecommendedLayerResolutionGetInfoMETA](#) structure. Application **may** choose to reallocate their swapchain or scale view resolution accordingly. Applications rendering multiple views into the swapchain **may** scale individual views to match the recommended swapchain resolution.

The runtime **may** not wish to make any recommendation, in which case it **must** return an [XrRecommendedLayerResolutionMETA::isValid](#) value of `XR_FALSE`.

If the [XrRecommendedLayerResolutionGetInfoMETA::layer](#) attribute of the **info** argument of the function contains valid swapchain handles in all fields where required, the runtime **must** return a resolution recommendation which is less than or equal to the size of that swapchain, so that the application **may** render into an existing swapchain or swapchains without reallocation. As an exception to valid usage, an otherwise-valid structure passed as [XrRecommendedLayerResolutionGetInfoMETA::layer](#) **may** contain `XR_NULL_HANDLE` in place of valid [XrSwapchain](#) handle(s) for this function only, to obtain a recommended resolution resolution for the purpose of allocating a swapchain. If at least one otherwise-required [XrSwapchain](#) handle within [XrRecommendedLayerResolutionGetInfoMETA::layer](#) is `XR_NULL_HANDLE`, the runtime **must** interpret this as a request for recommended resolution without limitation to the allocated size of any existing swapchain.

If the runtime makes a recommendation, it **should** make a recommendation that is directly usable by the application to render its frames without creating adverse visual effects for the user.

### Issues

1. Should this extension be leveraging events instead of being queried potentially every frame?

**RESOLVED:** Yes.

We want to provide the runtime the flexibility to smoothly transition the application from one resolution to another in a dynamic resolution usecase without any reallocation. To do so with an event system would send an event every frame which we preferred to avoid.

### Version History

- Revision 1, 2023-12-10 (Remi Palandri)
  - Initial extension description

## 12.99. XR\_META\_spatial\_entity\_mesh

### Name String

`XR_META_spatial_entity_mesh`

### Extension Type

Instance extension

### Registered Extension Number

270

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_FB\\_spatial\\_entity](#)

### Last Modified Date

2023-06-12

### IP Status

No known IP claims.

### Contributors

Yuichi Taguchi, Meta Platforms  
Anton Vaneev, Meta Platforms  
Andreas Loeve Selvik, Meta Platforms  
John Kearney, Meta Platforms

### 12.99.1. Overview

This extension expands on the concept of spatial entities to include a way for a spatial entity to represent a triangle mesh that describes 3D geometry of the spatial entity in a scene. Spatial entities are defined in [XR\\_FB\\_spatial\\_entity](#) extension using the Entity-Component System. The triangle mesh is a component type that **may** be associated to a spatial entity.

In order to enable the functionality of this extension, you **must** pass the name of the extension into [xrCreateInstance](#) via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

## 12.99.2. Retrieving a triangle mesh

The `xrGetSpaceTriangleMeshMETA` function is defined as:

```
// Provided by XR_META_spatial_entity_mesh
XrResult xrGetSpaceTriangleMeshMETA(
    XrSpace space,
    const XrSpaceTriangleMeshGetInfoMETA* getInfo,
    XrSpaceTriangleMeshMETA* triangleMeshOutput);
```

### Parameter Descriptions

- `space` is a handle to an [XrSpace](#).
- `getInfo` exists for extensibility purposes. It is `NULL` or a pointer to a valid [XrSpaceTriangleMeshGetInfoMETA](#).
- `triangleMeshOutput` is the output parameter that points to an [XrSpaceTriangleMeshMETA](#).

The `xrGetSpaceTriangleMeshMETA` function is used by the application to perform the two calls required to obtain a triangle mesh associated to a spatial entity specified by `space`.

The spatial entity `space` **must** have the `XR_SPACE_COMPONENT_TYPE_TRIANGLE_MESH_META` component type enabled, otherwise this function will return `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`.

### Valid Usage (Implicit)

- The `XR_META_spatial_entity_mesh` extension **must** be enabled prior to calling `xrGetSpaceTriangleMeshMETA`
- `space` **must** be a valid [XrSpace](#) handle
- `getInfo` **must** be a pointer to a valid [XrSpaceTriangleMeshGetInfoMETA](#) structure
- `triangleMeshOutput` **must** be a pointer to an [XrSpaceTriangleMeshMETA](#) structure



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SPACE_COMPONENT_NOT_ENABLED_FB`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrSpaceTriangleMeshGetInfoMETA` structure is defined as:

```
// Provided by XR_META_spatial_entity_mesh
typedef struct XrSpaceTriangleMeshGetInfoMETA {
    XrStructureType    type;
    const void*        next;
} XrSpaceTriangleMeshGetInfoMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

## Valid Usage (Implicit)

- The `XR_META_spatial_entity_mesh` extension **must** be enabled prior to using `XrSpaceTriangleMeshGetInfoMETA`
- `type` **must** be `XR_TYPE_SPACE_TRIANGLE_MESH_GET_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrSpaceTriangleMeshMETA` structure is defined as:

```
// Provided by XR_META_spatial_entity_mesh
typedef struct XrSpaceTriangleMeshMETA {
    XrStructureType    type;
    void*              next;
    uint32_t           vertexCapacityInput;
    uint32_t           vertexCountOutput;
    XrVector3f*        vertices;
    uint32_t           indexCapacityInput;
    uint32_t           indexCountOutput;
    uint32_t*          indices;
} XrSpaceTriangleMeshMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `vertexCapacityInput` is an input parameter for the application to specify the capacity of the `vertices` array, or 0 to indicate a request to retrieve the required capacity.
- `vertexCountOutput` is an output parameter that will hold the number of vertices written in the output array, or the required capacity in the case that `vertexCapacityInput` is insufficient. The returned value **must** be equal to or larger than 3.
- `vertices` is a pointer to an array of [XrVector3f](#), but **can** be `NULL` if `vertexCapacityInput` is 0. The vertices are defined in the coordinate frame of [XrSpace](#) to which this struct is associated.
- `indexCapacityInput` is an input parameter for the application to specify the capacity of the `indices` array, or 0 to indicate a request to retrieve the required capacity.
- `indexCountOutput` is an output parameter that will hold the number of indices written in the output array, or the required capacity in the case that `indexCapacityInput` is insufficient. The returned value **must** be a multiple of 3.
- `indices` is a pointer to an array of `uint32_t`, but **can** be `NULL` if `indexCapacityInput` is 0. Each element refers to a vertex in `vertices`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `vertices` and `indices` array sizes.

The [XrSpaceTriangleMeshMETA](#) structure **can** be used by the application to perform the two calls required to obtain a triangle mesh associated to a specified spatial entity.

The output values written in the `indices` array represent indices of vertices: Three consecutive elements represent a triangle with a counter-clockwise winding order.

## Valid Usage (Implicit)

- The `XR_META_spatial_entity_mesh` extension **must** be enabled prior to using [XrSpaceTriangleMeshMETA](#)
- `type` **must** be `XR_TYPE_SPACE_TRIANGLE_MESH_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Object Types

## New Atom

## New Flag Types

## New Enum Constants

[XrSpaceComponentTypeFB](#) enumeration is extended with:

- `XR_SPACE_COMPONENT_TYPE_TRIANGLE_MESH_META`

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SPACE_TRIANGLE_MESH_GET_INFO_META`
- `XR_TYPE_SPACE_TRIANGLE_MESH_META`

## New Enums

## New Structures

- [XrSpaceTriangleMeshGetInfoMETA](#)
- [XrSpaceTriangleMeshMETA](#)

## New Functions

- [xrGetSpaceTriangleMeshMETA](#)

## Issues

## Version History

- Revision 1, 2023-06-12 (Yuichi Taguchi)
  - Initial extension description.

# 12.100. XR\_META\_touch\_controller\_plus

## Name String

`XR_META_touch_controller_plus`

## Extension Type

Instance extension

## Registered Extension Number

280

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2023-04-10

## IP Status

No known IP claims.

## Contributors

Aanchal Dalmia, Meta Platforms

Adam Bengis, Meta Platforms

## Overview

This extension defines a new interaction profile for the Meta Quest Touch Plus Controller.

Meta Quest Touch Plus Controller interaction profile path:

- */interaction\_profiles/meta/touch\_controller\_plus*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile provides inputs and outputs that are a superset of those available in the existing "Oculus Touch Controller" interaction profile, */interaction\_profiles/oculus/touch\_controller*

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*

- *.../input/system/click* (**may** not be available for application use)
- On both:
  - *.../input/squeeze/value*
  - *.../input/trigger/value*
  - *.../input/trigger/touch*
  - *.../input/thumbstick*
  - *.../input/thumbstick/x*
  - *.../input/thumbstick/y*
  - *.../input/thumbstick/click*
  - *.../input/thumbstick/touch*
  - *.../input/thumbrest/touch*
  - *.../input/grip/pose*
  - *.../input/aim/pose*
  - *.../output/haptic*
  - *.../input/thumb\_meta/proximity\_meta*
  - *.../input/trigger/proximity\_meta*
  - *.../input/trigger/curl\_meta*
  - *.../input/trigger/slide\_meta*
  - *.../input/trigger/force*

## New Identifiers

- **thumb\_meta**: Meta Quest Touch Plus Controller adds an input identifier for the user's thumb on the same hand currently holding the controller. Thumb input is not explicitly bound to any location on the controller.

## Input Path Descriptions

- **/input/thumb\_meta/proximity\_meta** : Boolean indicating the user's thumb is near the inputs on the top face of the controller.
- **/input/trigger/proximity\_meta** : Boolean indicating whether the user's index finger is near the trigger.
- **/input/trigger/curl\_meta** : Float representing how pointed or curled the user's index finger is on the trigger: 0.0 = fully pointed, 1.0 = finger flat on the surface
- **/input/trigger/slide\_meta** : Float representing how far the user is sliding the tip of their index finger along the surface of the trigger: 0.0 = finger flat on the surface, 1.0 = finger fully drawn back.
- **/input/trigger/force** : Float representing the amount of force being applied by the user to the trigger after it reaches the end of the range of travel: 0.0 = no additional pressure applied, 1.0 = maximum detectable pressure applied.

### Version History

- Revision 1, 2023-04-10 (Adam Bengis)
  - Initial extension proposal

## 12.101. XR\_META\_virtual\_keyboard

### Name String

`XR_META_virtual_keyboard`

### Extension Type

Instance extension

### Registered Extension Number

220

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-04-14

### IP Status

No known IP claims.

## Contributors

Brent Housen, Meta Platforms  
Chiara Coetzee, Meta Platforms  
Juan Pablo León, Meta Platforms  
Peter Chan, Meta Platforms

## Contacts

Brent Housen, Meta Platforms  
Peter Chan, Meta Platforms

### 12.101.1. Overview

The virtual keyboard extension provides a system-driven localized keyboard that the application has full control over in terms of positioning and rendering.

This is achieved by giving the application the data required to drive rendering and animation of the keyboard in response to interaction data passed from the application to the runtime.

This approach is an alternative to a potential system keyboard overlay solution and provides a keyboard that can seamlessly blend into the application environment, since it is rendered by the same system, and avoids input focus issues that might come with a system overlay.

The API is also designed to work with custom hand and/or controller models in various games and applications.

### Virtual Keyboard Integration Summary

Before explaining the individual API functions, types, and events, here is an overview on how to integrate the virtual keyboard in an application.

Note that this is purely informational and does not serve as binding requirements for the runtime or the application.

#### *App Startup*

- Check if your device supports the virtual keyboard with [xrGetSystemProperties](#).
- Create a new keyboard with [xrCreateVirtualKeyboardMETA](#).
- Give it a location with [xrCreateVirtualKeyboardSpaceMETA](#), and keep a reference to the returned [XrSpace](#).
- Load the virtual keyboard glTF model using [XR\\_FB\\_render\\_model](#):
  - Query the render model key for path `/model_meta/keyboard/virtual`.
    - Using [xrEnumerateRenderModelPathsFB](#) and [xrGetRenderModelPropertiesFB](#).
    - Make sure to set the support level to `XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_2_BIT_FB`.
  - Load the render model glTF data with the given key with [xrLoadRenderModelFB](#).



- Load the glTF data into an extendable glTF renderer (see [Extend glTF render model support](#)). Note that this render model is hidden by default.

### Update Tick

- When the application wants to show the keyboard, call [xrSetVirtualKeyboardModelVisibilityMETA](#) to request the runtime to update the model visibility.
  - The application **should** wait for the [XrEventDataVirtualKeyboardShownMETA](#) event as confirmation that the runtime is ready to show the keyboard.
- The application **can** move the keyboard by calling [xrSuggestVirtualKeyboardLocationMETA](#) to update the saved [XrSpace](#).
- Then for every active input type feed the keyboard input with [xrSendVirtualKeyboardInputMETA](#):
  - For each hand/controller, use:
    - [XR\\_VIRTUAL\\_KEYBOARD\\_INPUT\\_SOURCE\\_\\*\\_RAY\\_\\*](#) for far input
    - [XR\\_VIRTUAL\\_KEYBOARD\\_INPUT\\_SOURCE\\_\\*\\_DIRECT\\_\\*](#) for direct/near input
    - If both near and far input types are sent, the runtime **may** decide which one is the most appropriate to use.
  - Passing in a value for the input devices interactorRoot as well, i.e. the wrist root for hands.
  - The runtime will modify the [interactorRootPose](#) to poke limit direct interaction.
    - If poke limiting is desired, the application **should** reposition input render models with the modified root pose.
- Then get the runtime keyboard pose and scale:
  - Using [xrLocateSpace](#) on the saved keyboardSpace.
  - Using [xrGetVirtualKeyboardScaleMETA](#) to get the scale.
- Then check if the virtual keyboard glTF model has any textures that need to be updated with [xrGetVirtualKeyboardDirtyTexturesMETA](#).
  - For every dirty texture, call [xrGetVirtualKeyboardTextureDataMETA](#) to get the RGBA texture data.
  - And then updating the texture in the glTF model that matches the given texture id.
- Then apply any glTF model animations using [xrGetVirtualKeyboardModelAnimationStatesMETA](#) to get updated animation indices and fraction values for each animation.

### On Events

- [XrEventDataVirtualKeyboardCommitTextMETA](#) / [XrEventDataVirtualKeyboardBackspaceMETA](#) / [XrEventDataVirtualKeyboardEnterMETA](#)
  - Applications **can** pipe these events to a focused input field, or whatever they are expecting to handle the virtual keyboard's input.
- [XrEventDataVirtualKeyboardShownMETA](#) & [XrEventDataVirtualKeyboardHiddenMETA](#)

- Signaled when the virtual keyboard render model animation system is hiding or showing the keyboard.

### *App Shutdown*

- Destroy the keyboard with [xrDestroyVirtualKeyboardMETA](#).

## 12.101.2. Extend glTF render model support

The virtual keyboard glTF model uses a custom texture URI for textures that the application needs to update dynamically. The application **should** implement a custom URI handler when loading the glTF model to check for these URIs and create writable textures identified by the corresponding texture ids.

The runtime **must** refer to these textures in the returned glTF model by URIs in the following format:

`metaVirtualKeyboard://texture/{textureID}?w={width}&h={height}&fmt=RGBA32`

The application **should** retrieve new pixel data from the runtime with [xrGetVirtualKeyboardDirtyTexturesMETA](#) and [xrGetVirtualKeyboardTextureDataMETA](#) and apply them to the corresponding textures that are used to render the glTF model.

Furthermore, the runtime **may** use additive morph target animations to control vertex coordinates and modify UVs. The application **should** check the "extras" property when loading a glTF animation channel for an integer field named "additiveWeightIndex". If present, this value indicates the morph target index that the animation weight should be applied to, or apply all weights if the value is -1.

The application **should** check for any glTF animations to apply to the model each frame with [xrGetVirtualKeyboardModelAnimationStatesMETA](#).

## 12.101.3. Collision Handling

Even though the runtime will handle any user interaction with the keyboard based on the input sent by the application, the application is responsible for managing how the keyboard should collide with other objects in the scene. To do this, the application **can** look for a node named "collision" in the loaded glTF model and use its mesh geometry and bound to define colliders that can be used by the application's choice of physics system.

## 12.101.4. Check device compatibility

When the [XR\\_META\\_virtual\\_keyboard](#) extension is enabled, an application **can** pass in an [XrSystemVirtualKeyboardPropertiesMETA](#) structure in the [XrSystemProperties::next](#) chain when calling [xrGetSystemProperties](#) to acquire information about the virtual keyboard's availability.

The [XrSystemVirtualKeyboardPropertiesMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrSystemVirtualKeyboardPropertiesMETA {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsVirtualKeyboard;
} XrSystemVirtualKeyboardPropertiesMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsVirtualKeyboard` is an [XrBool32](#) indicating if virtual keyboard is supported.

The struct is used for checking virtual keyboard support.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrSystemVirtualKeyboardPropertiesMETA](#)
- `type` **must** be `XR_TYPE_SYSTEM_VIRTUAL_KEYBOARD_PROPERTIES_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.101.5. Create a virtual keyboard

An application **can** create a virtual keyboard by calling [xrCreateVirtualKeyboardMETA](#).

The [xrCreateVirtualKeyboardMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrCreateVirtualKeyboardMETA(
    XrSession                session,
    const XrVirtualKeyboardCreateInfoMETA* createInfo,
    XrVirtualKeyboardMETA*   keyboard);
```

## Parameter Descriptions

- `session` is the [XrSession](#).
- `createInfo` is the [XrVirtualKeyboardCreateInfoMETA](#).
- `keyboard` is the returned [XrVirtualKeyboardMETA](#).

`xrCreateVirtualKeyboardMETA` creates an [XrVirtualKeyboardMETA](#) handle and establishes a keyboard within the runtime [XrSession](#). The returned virtual keyboard handle **may** be subsequently used in API calls.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to calling `xrCreateVirtualKeyboardMETA`
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrVirtualKeyboardCreateInfoMETA](#) structure
- `keyboard` **must** be a pointer to an [XrVirtualKeyboardMETA](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The [XrVirtualKeyboardCreateInfoMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardCreateInfoMETA {
    XrStructureType    type;
    const void*        next;
} XrVirtualKeyboardCreateInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

The struct is used for keyboard creation. Empty with the intention of future extension.

The runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if [XrSystemVirtualKeyboardPropertiesMETA::supportsVirtualKeyboard](#) is `XR_FALSE` when checking the device compatibility.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrVirtualKeyboardCreateInfoMETA](#)
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.101.6. Destroy the virtual keyboard

An application **can** destroy a virtual keyboard by calling [xrDestroyVirtualKeyboardMETA](#).

The [xrDestroyVirtualKeyboardMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrDestroyVirtualKeyboardMETA(
    XrVirtualKeyboardMETA          keyboard);
```

## Parameter Descriptions

- `keyboard` is the [XrVirtualKeyboardMETA](#) handle to the keyboard to destroy.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to calling [xrDestroyVirtualKeyboardMETA](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle

## Thread Safety

- Access to `keyboard`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## 12.101.7. Place the virtual keyboard

To place the keyboard, an application **can** create a virtual keyboard space by calling [xrCreateVirtualKeyboardSpaceMETA](#).

The [xrCreateVirtualKeyboardSpaceMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrCreateVirtualKeyboardSpaceMETA(
    XrSession session,
    XrVirtualKeyboardMETA keyboard,
    const XrVirtualKeyboardSpaceCreateInfoMETA* createInfo,
    XrSpace* keyboardSpace);
```

## Parameter Descriptions

- `session` is the [XrSession](#).
- `keyboard` is the [XrVirtualKeyboardMETA](#) handle.
- `createInfo` is the [XrVirtualKeyboardSpaceCreateInfoMETA](#).
- `keyboardSpace` is the returned space handle.

Creates an [XrSpace](#) handle and places the keyboard in this space. The returned space handle **may** be subsequently used in API calls.

Once placed, the application **should** query the keyboard's location each frame using [xrLocateSpace](#). It is important to do this every frame as the runtime is in control of the keyboard's movement.

The runtime **must** return `XR_ERROR_HANDLE_INVALID` if `session` is different than what is used to create `keyboard`.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to calling [xrCreateVirtualKeyboardSpaceMETA](#)
- `session` **must** be a valid [XrSession](#) handle
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle
- `createInfo` **must** be a pointer to a valid [XrVirtualKeyboardSpaceCreateInfoMETA](#) structure
- `keyboardSpace` **must** be a pointer to an [XrSpace](#) handle
- `keyboard` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_LIMIT\_REACHED
- XR\_ERROR\_POSE\_INVALID
- XR\_ERROR\_FEATURE\_UNSUPPORTED

The [XrVirtualKeyboardSpaceCreateInfoMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardSpaceCreateInfoMETA {
    XrStructureType          type;
    const void*              next;
    XrVirtualKeyboardLocationTypeMETA locationType;
    XrSpace                  space;
    XrPosef                  poseInSpace;
} XrVirtualKeyboardSpaceCreateInfoMETA;
```



## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `locationType` is an `XrVirtualKeyboardLocationTypeMETA` enum providing the location type.
- `space` is an `XrSpace` previously created by a function such as `xrCreateReferenceSpace`.
- `poseInSpace` is the desired pose if `locationType` is `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META`.

If `locationType` is set to `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META`, the runtime **must** use the value `poseInSpace` set by the application. Otherwise, the runtime **must** provide a default pose and ignore `poseInSpace`. In all cases, the runtime **must** default the scale to 1.0.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to using `XrVirtualKeyboardSpaceCreateInfoMETA`
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_SPACE_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `locationType` **must** be a valid `XrVirtualKeyboardLocationTypeMETA` value
- `space` **must** be a valid `XrSpace` handle

### 12.101.8. Move and scale the virtual keyboard

After creating a keyboard and a space, an application **can** request to move its location or change its scale. The application **can** suggest a new location or scale by calling `xrSuggestVirtualKeyboardLocationMETA`.

The `xrSuggestVirtualKeyboardLocationMETA` function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrSuggestVirtualKeyboardLocationMETA(
    XrVirtualKeyboardMETA          keyboard,
    const XrVirtualKeyboardLocationInfoMETA* locationInfo);
```

## Parameter Descriptions

- `keyboard` is the [XrVirtualKeyboardMETA](#) handle.
- `locationInfo` is the desired [XrVirtualKeyboardLocationInfoMETA](#).

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to calling [xrSuggestVirtualKeyboardLocationMETA](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle
- `locationInfo` **must** be a pointer to a valid [XrVirtualKeyboardLocationInfoMETA](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The [XrVirtualKeyboardLocationInfoMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardLocationInfoMETA {
    XrStructureType          type;
    const void*              next;
    XrVirtualKeyboardLocationTypeMETA  locationType;
    XrSpace                  space;
    XrPosef                  poseInSpace;
    float                    scale;
} XrVirtualKeyboardLocationInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `locationType` is an [XrVirtualKeyboardLocationTypeMETA](#) enum providing the location type.
- `space` is an [XrSpace](#) previously created by a function such as [xrCreateReferenceSpace](#).
- `poseInSpace` is the desired pose if `locationType` is `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META`.
- `scale` is a `float` value of the desired multiplicative scale between 0.0 and 1.0 if `locationType` is `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META`.

If `locationType` is set to `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META`, the runtime **must** use the values `poseInSpace` and `scale` set by the application. Otherwise, the runtime **must** provide a default pose and scale and ignore `poseInSpace` and `scale`.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to using [XrVirtualKeyboardLocationInfoMETA](#)
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_LOCATION_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `locationType` **must** be a valid [XrVirtualKeyboardLocationTypeMETA](#) value
- `space` **must** be a valid [XrSpace](#) handle

### 12.101.9. Get the virtual keyboard scale

Since [xrLocateSpace](#) only handles the pose, the application **should** also get the scale every frame by

calling [xrGetVirtualKeyboardScaleMETA](#).

The [xrGetVirtualKeyboardScaleMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrGetVirtualKeyboardScaleMETA(
    XrVirtualKeyboardMETA      keyboard,
    float*                      scale);
```

### Parameter Descriptions

- **keyboard** is the [XrVirtualKeyboardMETA](#) handle.
- **scale** is a float value of the current scale of the keyboard.

With both the pose and scale, the application has all the information to draw the virtual keyboard render model.

### Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to calling [xrGetVirtualKeyboardScaleMETA](#)
- **keyboard** **must** be a valid [XrVirtualKeyboardMETA](#) handle
- **scale** **must** be a pointer to a **float** value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

### 12.101.10. Show and hide the virtual keyboard

The runtime is in control of the keyboard's visibility to decide when to process input and reset the keyboard states. By default the keyboard render model is hidden. An application **can** update the render model visibility by calling `xrSetVirtualKeyboardModelVisibilityMETA`.

The `xrSetVirtualKeyboardModelVisibilityMETA` function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrSetVirtualKeyboardModelVisibilityMETA(
    XrVirtualKeyboardMETA          keyboard,
    const XrVirtualKeyboardModelVisibilitySetInfoMETA* modelVisibility);
```

### Parameter Descriptions

- `keyboard` is the `XrVirtualKeyboardMETA` handle.
- `modelVisibility` is the `XrVirtualKeyboardModelVisibilitySetInfoMETA`.

Note that the runtime has final control of the model visibility. The runtime **may** also change the visible state in certain situations. To get the actual visibility state of the render model, the application **should** wait for the `XrEventDataVirtualKeyboardShownMETA` and `XrEventDataVirtualKeyboardHiddenMETA` events.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to calling `xrSetVirtualKeyboardModelVisibilityMETA`
- `keyboard` **must** be a valid `XrVirtualKeyboardMETA` handle
- `modelVisibility` **must** be a pointer to a valid `XrVirtualKeyboardModelVisibilitySetInfoMETA` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrVirtualKeyboardModelVisibilitySetInfoMETA` structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardModelVisibilitySetInfoMETA {
    XrStructureType    type;
    const void*        next;
    XrBool32           visible;
} XrVirtualKeyboardModelVisibilitySetInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `visible` an [XrBool32](#) that controls whether to show or hide the keyboard.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrVirtualKeyboardModelVisibilitySetInfoMETA](#)
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_MODEL_VISIBILITY_SET_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.101.11. Update render model textures

Each frame update the application **should** check for any textures that are updated by the runtime (e.g. when new swipe suggestion words are available). The application **should** first get the texture IDs that have updated contents (are "dirty") by calling [xrGetVirtualKeyboardDirtyTexturesMETA](#). Then for each texture ID received, the application **should** create a [XrVirtualKeyboardTextureDataMETA](#) structure and call [xrGetVirtualKeyboardTextureDataMETA](#) to get the pixel data to update the corresponding texture created by the render system using the id reference.

The [xrGetVirtualKeyboardDirtyTexturesMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrGetVirtualKeyboardDirtyTexturesMETA(
    XrVirtualKeyboardMETA          keyboard,
    uint32_t                       textureIdCapacityInput,
    uint32_t*                      textureIdCountOutput,
    uint64_t*                      textureIds);
```

## Parameter Descriptions

- `keyboard` is the `XrVirtualKeyboardMETA` handle.
- `textureIdCapacityInput` is the capacity of the `textureIds` array, or 0 to indicate a request to retrieve the required capacity.
- `textureIdCountOutput` is filled in by the runtime with the count of texture IDs written or the required capacity in the case that `textureIdCapacityInput` is insufficient.
- `textureIds` is the array of texture IDs that need to be updated.

This function follows the [two-call idiom](#) for filling the `textureIds` array. Note that new texture data may be added after the runtime processes inputs from `xrSendVirtualKeyboardInputMETA`. Therefore, after sending new keyboard inputs the application **should** query the buffer size again before getting any texture data.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to calling `xrGetVirtualKeyboardDirtyTexturesMETA`
- `keyboard` **must** be a valid `XrVirtualKeyboardMETA` handle
- `textureIdCountOutput` **must** be a pointer to a `uint32_t` value
- If `textureIdCapacityInput` is not 0, `textureIds` **must** be a pointer to an array of `textureIdCapacityInput` `uint64_t` values



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrGetVirtualKeyboardTextureDataMETA` function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrGetVirtualKeyboardTextureDataMETA(
    XrVirtualKeyboardMETA          keyboard,
    uint64_t                       textureId,
    XrVirtualKeyboardTextureDataMETA* textureData);
```

## Parameter Descriptions

- `keyboard` is the `XrVirtualKeyboardMETA` handle.
- `textureId` is the ID of the texture that the application is querying data for.
- `textureData` is the returned `XrVirtualKeyboardTextureDataMETA`.

This function follows the [two-call idiom](#) for filling the `textureData` array in the `XrVirtualKeyboardTextureDataMETA` structure. Note that new texture data may be added after the runtime processes inputs from `xrSendVirtualKeyboardInputMETA`. Therefore, after sending new keyboard inputs the application **should** query the buffer size again before getting any texture data.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to calling `xrGetVirtualKeyboardTextureDataMETA`
- `keyboard` **must** be a valid `XrVirtualKeyboardMETA` handle
- `textureData` **must** be a pointer to an `XrVirtualKeyboardTextureDataMETA` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrVirtualKeyboardTextureDataMETA` structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardTextureDataMETA {
    XrStructureType    type;
    void*              next;
    uint32_t           textureWidth;
    uint32_t           textureHeight;
    uint32_t           bufferCapacityInput;
    uint32_t           bufferCountOutput;
    uint8_t*           buffer;
} XrVirtualKeyboardTextureDataMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `textureWidth` is the pixel width of the texture to be updated.
- `textureHeight` is the pixel height of the texture to be updated.
- `bufferCapacityInput` is the capacity of `buffer`, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is filled in by the runtime with the byte count written or the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is the pixel data in linear color space, RGBA 8-bit unsigned normalized integer format (i.e. `GL_RGBA8` in OpenGL, `VK_FORMAT_R8G8B8A8_UNORM` in Vulkan).

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrVirtualKeyboardTextureDataMETA](#)
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_TEXTURE_DATA_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` `uint8_t` values

### 12.101.12. Update render model animations

Besides checking for texture updates, each frame the application **should** also check for any animations to be applied to the render model. The runtime **may** use these animations to control the visibility of different keys, layout changes, and even modify key sizes and texture coordinates via morph targets. The application **can** get the animation states to be applied by calling [xrGetVirtualKeyboardModelAnimationStatesMETA](#). This will return an array of [XrVirtualKeyboardAnimationStateMETA](#) which the application **should** apply to the render model, indexed by the GLTF animation array index order.

The [xrGetVirtualKeyboardModelAnimationStatesMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrGetVirtualKeyboardModelAnimationStatesMETA(
    XrVirtualKeyboardMETA          keyboard,
    XrVirtualKeyboardModelAnimationStatesMETA* animationStates);
```

## Parameter Descriptions

- `keyboard` is the [XrVirtualKeyboardMETA](#) handle.
- `animationStates` is the [XrVirtualKeyboardModelAnimationStatesMETA](#).

This function follows the [two-call idiom](#) for filling the `animationStates` array in the [XrVirtualKeyboardModelAnimationStatesMETA](#) structure. Note that new animations may be added after the runtime processes inputs from [xrSendVirtualKeyboardInputMETA](#). Therefore, after sending new keyboard inputs the application **should** query the buffer size again before getting any animation data.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to calling [xrGetVirtualKeyboardModelAnimationStatesMETA](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle
- `animationStates` **must** be a pointer to an [XrVirtualKeyboardModelAnimationStatesMETA](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `XrVirtualKeyboardAnimationStateMETA` structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardAnimationStateMETA {
    XrStructureType    type;
    void*              next;
    int32_t            animationIndex;
    float              fraction;
} XrVirtualKeyboardAnimationStateMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `animationIndex` is the index of the animation to use for the render model.
- `fraction` is the normalized value between the start and end time of the animation.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to using `XrVirtualKeyboardAnimationStateMETA`
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_ANIMATION_STATE_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `XrVirtualKeyboardModelAnimationStatesMETA` structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardModelAnimationStatesMETA {
    XrStructureType          type;
    void*                    next;
    uint32_t                 stateCapacityInput;
    uint32_t                 stateCountOutput;
    XrVirtualKeyboardAnimationStateMETA* states;
} XrVirtualKeyboardModelAnimationStatesMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `stateCapacityInput` is the capacity of the `states` array, or 0 to indicate a request to retrieve the required capacity.
- `stateCountOutput` is filled in by the runtime with the count of [XrVirtualKeyboardAnimationStateMETA](#) written or the required capacity in the case that `stateCapacityInput` is insufficient.
- `states` is the array of [XrVirtualKeyboardAnimationStateMETA](#) to apply to the model.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to using `XrVirtualKeyboardModelAnimationStatesMETA`
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_MODEL_ANIMATION_STATES_META`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- If `stateCapacityInput` is not `0`, `states` **must** be a pointer to an array of `stateCapacityInput` `XrVirtualKeyboardAnimationStateMETA` structures

### 12.101.13. Send user input and text context

Since the application has control over how collision should be handled between the keyboard and other objects in the scene, it is up to the application to decide when to send input to the virtual keyboard. Per frame, for every input source the application wants to be applied to the keyboard, the application **should** create a `XrVirtualKeyboardInputInfoMETA` and call `xrSendVirtualKeyboardInputMETA` while also supplying the root pose of the interaction source.

The runtime **may** modify with an offset the given `interactorRootPose` if the given input is puncturing the keyboard. This is to give the effect that the virtual object cannot push through the keyboard and improves keyboard input perception. This is sometimes referred to as poke limiting.

To aid features like auto complete or whole word deletion, before sending input applications **should** populate a `XrVirtualKeyboardTextContextChangeInfoMETA` structure and call `xrChangeVirtualKeyboardTextContextMETA` to supply the runtime with the application's text context prior to the input cursor.

The `xrSendVirtualKeyboardInputMETA` function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrSendVirtualKeyboardInputMETA(
    XrVirtualKeyboardMETA          keyboard,
    const XrVirtualKeyboardInputInfoMETA* info,
    XrPosef*                       interactorRootPose);
```

## Parameter Descriptions

- `keyboard` is the `XrVirtualKeyboardMETA` handle.
- `info` is the `XrVirtualKeyboardInputInfoMETA` detailing the input being sent to the runtime.
- `interactorRootPose` is an `XrPosef` defining the root pose of the input source. The runtime **may** modify this value to aid keyboard input perception.

The application **can** use values like a pointer pose as the `interactorRootPose` for `XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_RAY_*` or `XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_RAY_*` input sources, a point on a controller model for `XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_DIRECT_*` input sources and the hand index tip pose for `XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_DIRECT_INDEX_TIP_*`. Different input poses can be used to accommodate application specific controller or hand models.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to calling `xrSendVirtualKeyboardInputMETA`
- `keyboard` **must** be a valid `XrVirtualKeyboardMETA` handle
- `info` **must** be a pointer to a valid `XrVirtualKeyboardInputInfoMETA` structure
- `interactorRootPose` **must** be a pointer to an `XrPosef` structure



## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_POSE\_INVALID
- XR\_ERROR\_FEATURE\_UNSUPPORTED

The [XrVirtualKeyboardInputInfoMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardInputInfoMETA {
    XrStructureType          type;
    const void*              next;
    XrVirtualKeyboardInputSourceMETA  inputSource;
    XrSpace                   inputSpace;
    XrPosef                   inputPoseInSpace;
    XrVirtualKeyboardInputStateFlagsMETA  inputState;
} XrVirtualKeyboardInputInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `inputSource` is an enum of [XrVirtualKeyboardInputSourceMETA](#) describing the source device and input mode type.
- `inputSpace` is an [XrSpace](#) previously created by a function such as [xrCreateReferenceSpace](#).
- `inputPoseInSpace` is an [XrPosef](#) defining the position and orientation of the input's source pose within the natural reference frame of the input space.
- `inputState` is a bitmask of [XrVirtualKeyboardInputStateFlagsMETA](#) describing the button or pinch state of the `inputSource`.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrVirtualKeyboardInputInfoMETA](#)
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_INPUT_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `inputSource` **must** be a valid [XrVirtualKeyboardInputSourceMETA](#) value
- If `inputSpace` is not `XR_NULL_HANDLE`, `inputSpace` **must** be a valid [XrSpace](#) handle
- `inputState` **must** be `0` or a valid combination of [XrVirtualKeyboardInputStateFlagBitsMETA](#) values

The [xrChangeVirtualKeyboardTextContextMETA](#) function is defined as:

```
// Provided by XR_META_virtual_keyboard
XrResult xrChangeVirtualKeyboardTextContextMETA(
    XrVirtualKeyboardMETA          keyboard,
    const XrVirtualKeyboardTextContextChangeInfoMETA* changeInfo);
```

## Parameter Descriptions

- `keyboard` is the [XrVirtualKeyboardMETA](#) handle.
- `changeInfo` is the [XrVirtualKeyboardTextContextChangeInfoMETA](#) detailing prior input text context to the runtime.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to calling [xrChangeVirtualKeyboardTextContextMETA](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle
- `changeInfo` **must** be a pointer to a valid [XrVirtualKeyboardTextContextChangeInfoMETA](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The [XrVirtualKeyboardTextContextChangeInfoMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrVirtualKeyboardTextContextChangeInfoMETA {
    XrStructureType    type;
    const void*        next;
    const char*        textContext;
} XrVirtualKeyboardTextContextChangeInfoMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `textContext` is a pointer to a `char` buffer, should contain prior input text context terminated with a null character.

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrVirtualKeyboardTextContextChangeInfoMETA](#)
- `type` **must** be `XR_TYPE_VIRTUAL_KEYBOARD_TEXT_CONTEXT_CHANGE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `textContext` **must** be a null-terminated UTF-8 string

### 12.101.14. Handling events

Each frame the application **should** listen for the following events sent by the runtime that reflects the state of the keyboard.

The [XrEventDataVirtualKeyboardCommitTextMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrEventDataVirtualKeyboardCommitTextMETA {
    XrStructureType    type;
    const void*        next;
    XrVirtualKeyboardMETA    keyboard;
    char                text[XR_MAX_VIRTUAL_KEYBOARD_COMMIT_TEXT_SIZE_META];
} XrEventDataVirtualKeyboardCommitTextMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `keyboard` is the [XrVirtualKeyboardMETA](#) this event belongs to.
- `text` is the text string input by the keyboard.

The [XrEventDataVirtualKeyboardCommitTextMETA](#) event **must** be sent by the runtime when a character or string is input by the keyboard. The application **should** append to the text field that the keyboard is editing.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to using [XrEventDataVirtualKeyboardCommitTextMETA](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_COMMIT_TEXT_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle
- `text` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_VIRTUAL_KEYBOARD_COMMIT_TEXT_SIZE_META`

The [XrEventDataVirtualKeyboardBackspaceMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrEventDataVirtualKeyboardBackspaceMETA {
    XrStructureType      type;
    const void*          next;
    XrVirtualKeyboardMETA keyboard;
} XrEventDataVirtualKeyboardBackspaceMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `keyboard` is the [XrVirtualKeyboardMETA](#) this event belongs to.

The [XrEventDataVirtualKeyboardBackspaceMETA](#) event **must** be sent by the runtime when the [Backspace] key is pressed. The application **should** update the text field that the keyboard is editing.

## Valid Usage (Implicit)

- The `XR_META_virtual_keyboard` extension **must** be enabled prior to using [XrEventDataVirtualKeyboardBackspaceMETA](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_BACKSPACE_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle

The [XrEventDataVirtualKeyboardEnterMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrEventDataVirtualKeyboardEnterMETA {
    XrStructureType      type;
    const void*          next;
    XrVirtualKeyboardMETA keyboard;
} XrEventDataVirtualKeyboardEnterMETA;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `keyboard` is the [XrVirtualKeyboardMETA](#) this event belongs to.

The [XrEventDataVirtualKeyboardEnterMETA](#) event **must** be sent by the runtime when the [Enter] key is pressed. The application **should** respond accordingly (e.g. newline, accept, etc).

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrEventDataVirtualKeyboardEnterMETA](#)
- **type** **must** be `XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_ENTER_META`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **keyboard** **must** be a valid [XrVirtualKeyboardMETA](#) handle

The [XrEventDataVirtualKeyboardShownMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrEventDataVirtualKeyboardShownMETA {
    XrStructureType      type;
    const void*         next;
    XrVirtualKeyboardMETA keyboard;
} XrEventDataVirtualKeyboardShownMETA;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **keyboard** is the [XrVirtualKeyboardMETA](#) this event belongs to.

The [XrEventDataVirtualKeyboardShownMETA](#) event **must** be sent when the runtime has shown the keyboard render model (via animation). The application **should** update its state accordingly (e.g. update UI, pause simulation, etc).

## Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrEventDataVirtualKeyboardShownMETA](#)
- **type** **must** be `XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_SHOWN_META`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **keyboard** **must** be a valid [XrVirtualKeyboardMETA](#) handle

The [XrEventDataVirtualKeyboardHiddenMETA](#) structure is defined as:

```
// Provided by XR_META_virtual_keyboard
typedef struct XrEventDataVirtualKeyboardHiddenMETA {
    XrStructureType      type;
    const void*         next;
    XrVirtualKeyboardMETA keyboard;
} XrEventDataVirtualKeyboardHiddenMETA;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `keyboard` is the [XrVirtualKeyboardMETA](#) this event belongs to.

The [XrEventDataVirtualKeyboardHiddenMETA](#) event **must** be sent when the keyboard render model is hidden by the runtime (via animation). The application **should** update its state accordingly (e.g. update UI, resume simulation, etc).

### Valid Usage (Implicit)

- The [XR\\_META\\_virtual\\_keyboard](#) extension **must** be enabled prior to using [XrEventDataVirtualKeyboardHiddenMETA](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_HIDDEN_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `keyboard` **must** be a valid [XrVirtualKeyboardMETA](#) handle

## 12.101.15. Example code for using virtual keyboard

The following example code demonstrates how to create and use the virtual keyboard.

```
XrInstance instance; // previously initialized
XrSystemId system;   // previously initialized
XrSession session;   // previously initialized
XrSpace localSpace;  // previously initialized
XrPosef poseIdentity; // previously initialized
```



```

// XR_FB_render_model API previously initialized with xrGetInstanceProcAddr
PFN_xrEnumerateRenderModelPathsFB xrEnumerateRenderModelPathsFB;
PFN_xrGetRenderModelPropertiesFB xrGetRenderModelPropertiesFB;
PFN_xrLoadRenderModelFB xrLoadRenderModelFB;

// XR_META_virtual_keyboard API previously initialized with xrGetInstanceProcAddr
PFN_xrCreateVirtualKeyboardMETA xrCreateVirtualKeyboardMETA;
PFN_xrDestroyVirtualKeyboardMETA xrDestroyVirtualKeyboardMETA;
PFN_xrCreateVirtualKeyboardSpaceMETA xrCreateVirtualKeyboardSpaceMETA;
PFN_xrSuggestVirtualKeyboardLocationMETA xrSuggestVirtualKeyboardLocationMETA;
PFN_xrGetVirtualKeyboardScaleMETA xrGetVirtualKeyboardScaleMETA;
PFN_xrSetVirtualKeyboardModelVisibilityMETA xrSetVirtualKeyboardModelVisibilityMETA;
PFN_xrGetVirtualKeyboardModelAnimationStatesMETA
xrGetVirtualKeyboardModelAnimationStatesMETA;
PFN_xrGetVirtualKeyboardDirtyTexturesMETA xrGetVirtualKeyboardDirtyTexturesMETA;
PFN_xrGetVirtualKeyboardTextureDataMETA xrGetVirtualKeyboardTextureDataMETA;
PFN_xrSendVirtualKeyboardInputMETA xrSendVirtualKeyboardInputMETA;

XrVirtualKeyboardMETA keyboardHandle{XR_NULL_HANDLE};
XrSpace keyboardSpace{XR_NULL_HANDLE};
XrRenderModelKeyFB keyboardModelKey{XR_NULL_RENDER_MODEL_KEY_FB};

// Check virtual keyboard support
XrSystemVirtualKeyboardPropertiesMETA
virtualKeyboardProps{XR_TYPE_SYSTEM_VIRTUAL_KEYBOARD_PROPERTIES_META};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES, &virtualKeyboardProps};
CHK_XR(xrGetSystemProperties(instance, system, &systemProperties));
if (virtualKeyboardProps.supportsVirtualKeyboard == XR_FALSE) {
    return; // Virtual keyboard not supported
}

// Create virtual keyboard and space
XrVirtualKeyboardCreateInfoMETA createInfo{XR_TYPE_VIRTUAL_KEYBOARD_CREATE_INFO_META};
CHK_XR(xrCreateVirtualKeyboardMETA(session, &createInfo, &keyboardHandle));

XrVirtualKeyboardSpaceCreateInfoMETA
spaceCreateInfo{XR_TYPE_VIRTUAL_KEYBOARD_SPACE_CREATE_INFO_META};
spaceCreateInfo.locationType = XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META;
spaceCreateInfo.space = localSpace;
spaceCreateInfo.poseInSpace = poseIdentity;
CHK_XR(xrCreateVirtualKeyboardSpaceMETA(session, keyboardHandle, &spaceCreateInfo,
&keyboardSpace));

// Get render model key
uint32_t pathCount = 0;
CHK_XR(xrEnumerateRenderModelPathsFB(session, pathCount, &pathCount, nullptr));
std::vector<XrRenderModelPathInfoFB> pathInfos(pathCount,
{XR_TYPE_RENDER_MODEL_PATH_INFO_FB});

```

```

CHK_XR(xrEnumerateRenderModelPathsFB(session, pathCount, &pathCount, pathInfos.data()));

for (const auto& info : pathInfos) {
    char pathString[XR_MAX_PATH_LENGTH];
    uint32_t countOutput = 0;
    CHK_XR(xrPathToString(instance, info.path, XR_MAX_PATH_LENGTH, &countOutput,
pathString));

    if (strcmp(pathString, "/model_meta/keyboard/virtual") == 0) {
        XrRenderModelPropertiesFB prop{XR_TYPE_RENDER_MODEL_PROPERTIES_FB};
        XrRenderModelCapabilitiesRequestFB
capReq{XR_TYPE_RENDER_MODEL_CAPABILITIES_REQUEST_FB};
        capReq.flags = XR_RENDER_MODEL_SUPPORTS_GLTF_2_0_SUBSET_2_BIT_FB;
        prop.next = &capReq;
        CHK_XR(xrGetRenderModelPropertiesFB(session, info.path, &prop));
        keyboardModelKey = prop.modelKey;
        break;
    }
}

if (keyboardModelKey == XR_NULL_RENDER_MODEL_KEY_FB) {
    return; // Model not available
}

/// Load render model
XrRenderModelLoadInfoFB loadInfo{XR_TYPE_RENDER_MODEL_LOAD_INFO_FB};
loadInfo.modelKey = keyboardModelKey;
XrRenderModelBufferFB renderModelbuffer{XR_TYPE_RENDER_MODEL_BUFFER_FB};
CHK_XR((xrLoadRenderModelFB(session, &loadInfo, &renderModelbuffer)));
std::vector<uint8_t> modelBuffer(renderModelbuffer.bufferCountOutput);
renderModelbuffer.buffer = modelBuffer.data();
renderModelbuffer.bufferCapacityInput = renderModelbuffer.bufferCountOutput;
CHK_XR((xrLoadRenderModelFB(session, &loadInfo, &renderModelbuffer)));
// >>> Application loads the glTF model in `modelBuffer`, keeping a reference to the
model animations and any textures with a URI texture id. See `Extend glTF render model
support`.

/// Show render model
XrVirtualKeyboardModelVisibilitySetInfoMETA
modelVisibility{XR_TYPE_VIRTUAL_KEYBOARD_MODEL_VISIBILITY_SET_INFO_META};
modelVisibility.visible = XR_TRUE;
CHK_XR(xrSetVirtualKeyboardModelVisibilityMETA(keyboardHandle, &modelVisibility));

while (!quit) {
    // ...
    // For every frame in frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame

```

```

const XrTime time = frameState.predictedDisplayTime;

XrVirtualKeyboardLocationInfoMETA
locationInfo{XR_TYPE_VIRTUAL_KEYBOARD_LOCATION_INFO_META};
// >>> Application sets desired location and scale in `locationInfo`
CHK_XR(xrSuggestVirtualKeyboardLocationMETA(keyboardHandle, &locationInfo));

// For each input source:
{
    XrVirtualKeyboardInputInfoMETA inputInfo{XR_TYPE_VIRTUAL_KEYBOARD_INPUT_INFO_META};
    // >>> Application sets input source data in `inputInfo`
    XrPosef interactorRootPose;
    CHK_XR(xrSendVirtualKeyboardInputMETA(keyboardHandle, &inputInfo,
&interactorRootPose));
    // >>> Application uses `interactorRootPose` as feedback for poke limiting
}

uint32_t textureIdCountOutput = 0;
CHK_XR(xrGetVirtualKeyboardDirtyTexturesMETA(keyboardHandle, 0, &textureIdCountOutput,
nullptr));
std::vector<uint64_t> dirtyTextureIds(textureIdCountOutput);
CHK_XR(xrGetVirtualKeyboardDirtyTexturesMETA(keyboardHandle, textureIdCountOutput,
&textureIdCountOutput, dirtyTextureIds.data()));
for (const uint64_t textureId : dirtyTextureIds) {
    XrVirtualKeyboardTextureDataMETA
textureData{XR_TYPE_VIRTUAL_KEYBOARD_TEXTURE_DATA_META};
    CHK_XR(xrGetVirtualKeyboardTextureDataMETA(keyboardHandle, textureId, &textureData));
    std::vector<uint8_t> textureDataBuffer(textureData.bufferCountOutput);
    textureData.bufferCapacityInput = textureData.bufferCountOutput;
    textureData.buffer = textureDataBuffer.data();
    CHK_XR(xrGetVirtualKeyboardTextureDataMETA(keyboardHandle, textureId, &textureData));
    // >>> Application applies `textureData` to the glTF texture referenced by
`textureId`
}

XrVirtualKeyboardModelAnimationStatesMETA
animationStates{XR_TYPE_VIRTUAL_KEYBOARD_MODEL_ANIMATION_STATES_META};
    CHK_XR(xrGetVirtualKeyboardModelAnimationStatesMETA(keyboardHandle, &animationStates));
    std::vector<XrVirtualKeyboardAnimationStateMETA>
animationStatesBuffer(animationStates.stateCountOutput,
{XR_TYPE_VIRTUAL_KEYBOARD_ANIMATION_STATE_META});
    animationStates.stateCapacityInput = animationStates.stateCountOutput;
    animationStates.states = animationStatesBuffer.data();
    CHK_XR(xrGetVirtualKeyboardModelAnimationStatesMETA(keyboardHandle, &animationStates));
    for (uint32_t i = 0; i < animationStates.stateCountOutput; ++i) {
        const auto& animationState = animationStates.states[i];
        // >>> Application applies `animationState` to the corresponding glTF model animation
    }
}

```

```

XrSpaceLocation keyboardLocation{XR_TYPE_SPACE_LOCATION};
CHK_XR(xrLocateSpace(keyboardSpace, localSpace, time, &keyboardLocation));
float keyboardScale;
CHK_XR(xrGetVirtualKeyboardScaleMETA(keyboardHandle, &keyboardScale));
// >>> Application renders model with `keyboardLocation` and `keyboardScale`
}

CHK_XR(xrDestroyVirtualKeyboardMETA(keyboardHandle));

```

## New Object Types

```
XR_DEFINE_HANDLE(XrVirtualKeyboardMETA)
```

[XrVirtualKeyboardMETA](#) represents a virtual keyboard instance.

## New Flag Types

```
typedef XrFlags64 XrVirtualKeyboardInputStateFlagsMETA;
```

```

// Flag bits for XrVirtualKeyboardInputStateFlagsMETA
static const XrVirtualKeyboardInputStateFlagsMETA
XR_VIRTUAL_KEYBOARD_INPUT_STATE_PRESSED_BIT_META = 0x00000001;

```

## Flag Descriptions

- `XR_VIRTUAL_KEYBOARD_INPUT_STATE_PRESSED_BIT_META` — If the input source is considered 'pressed' at all. Pinch for hands, Primary button for controllers.

## New Enum Constants

- `XR_MAX_VIRTUAL_KEYBOARD_COMMIT_TEXT_SIZE_META`

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SYSTEM_VIRTUAL_KEYBOARD_PROPERTIES_META`

- XR\_TYPE\_VIRTUAL\_KEYBOARD\_CREATE\_INFO\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_SPACE\_CREATE\_INFO\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_LOCATION\_INFO\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_MODEL\_VISIBILITY\_SET\_INFO\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_ANIMATION\_STATE\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_MODEL\_ANIMATION\_STATES\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_TEXTURE\_DATA\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_INPUT\_INFO\_META
- XR\_TYPE\_VIRTUAL\_KEYBOARD\_TEXT\_CONTEXT\_CHANGE\_INFO\_META
- XR\_TYPE\_EVENT\_DATA\_VIRTUAL\_KEYBOARD\_COMMIT\_TEXT\_META
- XR\_TYPE\_EVENT\_DATA\_VIRTUAL\_KEYBOARD\_BACKSPACE\_META
- XR\_TYPE\_EVENT\_DATA\_VIRTUAL\_KEYBOARD\_ENTER\_META
- XR\_TYPE\_EVENT\_DATA\_VIRTUAL\_KEYBOARD\_SHOWN\_META
- XR\_TYPE\_EVENT\_DATA\_VIRTUAL\_KEYBOARD\_HIDDEN\_META

## New Defines

## New Enums

The possible location types are specified by the [XrVirtualKeyboardLocationTypeMETA](#) enumeration:

```
// Provided by XR_META_virtual_keyboard
typedef enum XrVirtualKeyboardLocationTypeMETA {
    XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META = 0,
    XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_FAR_META = 1,
    XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_DIRECT_META = 2,
    XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_MAX_ENUM_META = 0x7FFFFFFF
} XrVirtualKeyboardLocationTypeMETA;
```

## Enumerant Descriptions

- `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_CUSTOM_META`  
Indicates that the application will provide the position and scale of the keyboard.
- `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_FAR_META`  
Indicates that the runtime will set the position and scale for far field keyboard.
- `XR_VIRTUAL_KEYBOARD_LOCATION_TYPE_DIRECT_META`  
Indicates that the runtime will set the position and scale for direct interaction keyboard.

The possible input sources are specified by the `XrVirtualKeyboardInputSourceMETA` enumeration:

```
// Provided by XR_META_virtual_keyboard
typedef enum XrVirtualKeyboardInputSourceMETA {
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_RAY_LEFT_META = 1,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_RAY_RIGHT_META = 2,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_RAY_LEFT_META = 3,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_RAY_RIGHT_META = 4,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_DIRECT_LEFT_META = 5,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_DIRECT_RIGHT_META = 6,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_DIRECT_INDEX_TIP_LEFT_META = 7,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_DIRECT_INDEX_TIP_RIGHT_META = 8,
    XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_MAX_ENUM_META = 0x7FFFFFFF
} XrVirtualKeyboardInputSourceMETA;
```

| Enum  | Description                    |
|---|--------------------------------|
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_RAY_LEFT_META</code>        | Left controller ray.           |
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_RAY_RIGHT_META</code>       | Right controller ray.          |
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_RAY_LEFT_META</code>              | Left hand ray.                 |
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_RAY_RIGHT_META</code>             | Right hand ray.                |
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_DIRECT_LEFT_META</code>     | Left controller direct touch.  |
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_CONTROLLER_DIRECT_RIGHT_META</code>    | Right controller direct touch. |
| <code>XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_DIRECT_INDEX_TIP_LEFT_META</code> | Left hand direct touch.        |

| Enum   | Description              |
|--|--------------------------|
| <a href="#">XR_VIRTUAL_KEYBOARD_INPUT_SOURCE_HAND_DIRECT_INDEXTIP_RIGHT_META</a> | Right hand direct touch. |

## New Structures

- [XrSystemVirtualKeyboardPropertiesMETA](#)
- [XrVirtualKeyboardCreateInfoMETA](#)
- [XrVirtualKeyboardSpaceCreateInfoMETA](#)
- [XrVirtualKeyboardLocationInfoMETA](#)
- [XrVirtualKeyboardModelVisibilitySetInfoMETA](#)
- [XrVirtualKeyboardAnimationStateMETA](#)
- [XrVirtualKeyboardModelAnimationStatesMETA](#)
- [XrVirtualKeyboardTextureDataMETA](#)
- [XrVirtualKeyboardInputInfoMETA](#)
- [XrVirtualKeyboardTextContextChangeInfoMETA](#)
- [XrEventDataVirtualKeyboardCommitTextMETA](#)
- [XrEventDataVirtualKeyboardBackspaceMETA](#)
- [XrEventDataVirtualKeyboardEnterMETA](#)
- [XrEventDataVirtualKeyboardShownMETA](#)
- [XrEventDataVirtualKeyboardHiddenMETA](#)

## New Functions

- [xrCreateVirtualKeyboardMETA](#)
- [xrDestroyVirtualKeyboardMETA](#)
- [xrCreateVirtualKeyboardSpaceMETA](#)
- [xrSuggestVirtualKeyboardLocationMETA](#)
- [xrGetVirtualKeyboardScaleMETA](#)
- [xrSetVirtualKeyboardModelVisibilityMETA](#)
- [xrGetVirtualKeyboardModelAnimationStatesMETA](#)
- [xrGetVirtualKeyboardDirtyTexturesMETA](#)
- [xrGetVirtualKeyboardTextureDataMETA](#)
- [xrSendVirtualKeyboardInputMETA](#)
- [xrChangeVirtualKeyboardTextContextMETA](#)

## Issues

## Version History

- Revision 1, 2023-04-14 (Peter Chan, Brent Housen)
  - Initial extension description

# 12.102. XR\_META\_vulkan\_swapchain\_create\_info

## Name String

`XR_META_vulkan_swapchain_create_info`

## Extension Type

Instance extension

## Registered Extension Number

228

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-05-19

## IP Status

No known IP claims.

## Contributors

John Kearney, Meta Platforms  
Andreas L. Selvik, Meta Platforms  
Jakob Bornecrantz, Collabora  
Ross Ning, Meta Platforms

## Overview

Using this extension, a Vulkan-based application **can** pass through additional `VkImageCreateFlags` or `VkImageUsageFlags` by chaining an `XrVulkanSwapchainCreateInfoMETA` structure to the `XrSwapchainCreateInfo` when calling `xrCreateSwapchain`.

The application is still encouraged to use the common bits like `XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT` defined in `XrSwapchainUsageFlags`. However, the application **may** present both `XR_SWAPCHAIN_USAGE_TRANSFER_SRC_BIT` in `XrSwapchainUsageFlags` and `VK_IMAGE_USAGE_TRANSFER_SRC_BIT`



in `VkImageUsageFlags` at the same time.

The application **must** enable the corresponding Vulkan extensions before requesting additional Vulkan flags. For example, `VK_EXT_fragment_density_map` device extension **must** be enabled if an application requests `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT` bit. Otherwise, it **may** cause undefined behavior, including an application crash.

Runtimes that implement this extension **must** support the `XR_KHR_vulkan_enable` or the `XR_KHR_vulkan_enable2` extension.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

```
// Provided by XR_META_vulkan_swapchain_create_info
typedef struct XrVulkanSwapchainCreateInfoMETA {
    XrStructureType      type;
    const void*          next;
    VkImageCreateFlags   additionalCreateFlags;
    VkImageUsageFlags    additionalUsageFlags;
} XrVulkanSwapchainCreateInfoMETA;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `additionalCreateFlags` is a bitmask of `VkImageCreateFlags` describing additional parameters of an image.
- `additionalUsageFlags` is a bitmask of `VkImageUsageFlags` describing additional parameters of an image.

The runtime **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if any bit of either `additionalCreateFlags` or `additionalUsageFlags` is not supported.

## Valid Usage (Implicit)

- The `XR_META_vulkan_swapchain_create_info` extension **must** be enabled prior to using `XrVulkanSwapchainCreateInfoMETA`
- `type` **must** be `XR_TYPE_VULKAN_SWAPCHAIN_CREATE_INFO_META`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `additionalCreateFlags` **must** be a valid `VkImageCreateFlags` value
- `additionalUsageFlags` **must** be a valid `VkImageUsageFlags` value

### New Functions

### Issues

### Version History

- Revision 1, 2022-05-05 (Ross Ning)
  - Initial draft

## 12.103. XR\_ML\_compat

### Name String

`XR_ML_compat`

### Extension Type

Instance extension

### Registered Extension Number

138

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-11-08

### Contributors

Ron Bessems, Magic Leap

### Overview

This extension provides functionality to facilitate transitioning from Magic Leap SDK to OpenXR SDK, most notably interoperability between Coordinate Frame UUIDs and [XrSpace](#).

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_COORDINATE_SPACE_CREATE_INFO_ML`

## New Structures

The [XrCoordinateSpaceCreateInfoML](#) structure is defined as:

```
typedef struct XrCoordinateSpaceCreateInfoML {
    XrStructureType    type;
    const void*        next;
    MLCoordinateFrameUID cfuid;
    XrPosef            poseInCoordinateSpace;
} XrCoordinateSpaceCreateInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `cfuid` is the [MLCoordinateFrameUID](#) as generated by the non-OpenXR API in the Magic Leap SDK.
- `poseInCoordinateSpace` is an [XrPosef](#) defining the position and orientation of the new space's origin within the natural reference frame of the `cfuid`.

[XrCoordinateSpaceCreateInfoML](#) is provided as input when calling [xrCreateSpaceFromCoordinateFrameUIDML](#) to convert a Magic Leap SDK generated [MLCoordinateFrameUID](#) to an [XrSpace](#). The conversion only needs to be done once even if the underlying [MLCoordinateFrameUID](#) changes its pose.

## Valid Usage (Implicit)

- The `XR_ML_compat` extension **must** be enabled prior to using `XrCoordinateSpaceCreateInfoML`
- `type` **must** be `XR_TYPE_COORDINATE_SPACE_CREATE_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `cfuid` **must** be a valid `MLCoordinateFrameUID` value

## New Functions

The `xrCreateSpaceFromCoordinateFrameUIDML` function is defined as:

```
// Provided by XR_ML_compat
XrResult xrCreateSpaceFromCoordinateFrameUIDML(
    XrSession session,
    const XrCoordinateSpaceCreateInfoML * createInfo,
    XrSpace* space);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession` previously created with `xrCreateSession`.
- `createInfo` is the `XrCoordinateSpaceCreateInfoML` used to specify the space.
- `space` is the returned space handle.

The service that created the underlying `XrCoordinateSpaceCreateInfoML::cfuid` **must** remain active for the lifetime of the `XrSpace`. If `xrLocateSpace` is called on a space created from an `XrCoordinateSpaceCreateInfoML::cfuid` from a no-longer-active service, the runtime **may** set `XrSpaceLocation::locationFlags` to 0.

`XrSpace` handles are destroyed using `xrDestroySpace`.

## Valid Usage (Implicit)

- The `XR ML_compat` extension **must** be enabled prior to calling `xrCreateSpaceFromCoordinateFrameUIDML`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrCoordinateSpaceCreateInfoML` structure
- `space` **must** be a pointer to an `XrSpace` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`

## Issues

## Version History

- Revision 1, 2022-11-08 (Ron Bessems)
  - Initial extension description

## 12.104. XR ML\_frame\_end\_info

### Name String

`XR ML_frame_end_info`

## Extension Type

Instance extension

## Registered Extension Number

136

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2022-10-26

## Contributors

Ron Bessems, Magic Leap

## Overview

This extension provides access to Magic Leap specific extensions to frame settings like focus distance, vignette, and protection.

## New Flag Types

The `XrFrameEndInfoML::flags` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in `XrFrameEndInfoFlagBitsML`.

```
typedef XrFlags64 XrFrameEndInfoFlagsML;
```

Valid bits for `XrFrameEndInfoFlagsML` are defined by `XrFrameEndInfoFlagBitsML`, which is specified as:

```
// Flag bits for XrFrameEndInfoFlagsML
static const XrFrameEndInfoFlagsML XR_FRAME_END_INFO_PROTECTED_BIT_ML = 0x00000001;
static const XrFrameEndInfoFlagsML XR_FRAME_END_INFO_VIGNETTE_BIT_ML = 0x00000002;
```

The flag bits have the following meanings:

## Flag Descriptions

- `XR_FRAME_END_INFO_PROTECTED_BIT_ML` — Indicates that the content for this frame is protected and should not be recorded or captured outside the graphics system.
- `XR_FRAME_END_INFO_VIGNETTE_BIT_ML` — Indicates that a soft fade to transparent should be added to the frame in the compositor to blend any hard edges at the FOV limits.

### New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_FRAME_END_INFO_ML`

### New Structures

The `XrFrameEndInfoML` structure is defined as:

```
// Provided by XR_ML_frame_end_info
typedef struct XrFrameEndInfoML {
    XrStructureType      type;
    const void*         next;
    float                focusDistance;
    XrFrameEndInfoFlagsML flags;
} XrFrameEndInfoML;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `focusDistance` is the distance, in meters, to defined focus point for the client content. The focus distance is interpreted as the positive distance to the client-determined object of interest (relative to the forward vector of the Lightwear).
- `flags` is a bitmask of `XrFrameEndInfoFlagsML`

## Valid Usage (Implicit)

- The `XR_ML_frame_end_info` extension **must** be enabled prior to using `XrFrameEndInfoML`
- `type` **must** be `XR_TYPE_FRAME_END_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be `0` or a valid combination of `XrFrameEndInfoFlagBitsML` values

### Version History

- Revision 1, 2022-10-26 (Ron Bessems)
  - Initial extension description

## 12.105. XR\_ML\_global\_dimmer

### Name String

`XR_ML_global_dimmer`

### Extension Type

Instance extension

### Registered Extension Number

137

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-10-25

### Contributors

Ron Bessems, Magic Leap  
Michał Kulągowski, Magic Leap

### Overview

This extension provides control over the global dimmer panel of the Magic Leap 2. The Global Dimming™ feature dims the entire display without dimming digital content to make text and images more solid and precise.

Note that when using the `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` mode the alpha channel of the color



swapchain image is combined with the global dimmer value. The global dimmer however is able to address the whole panel whereas the alpha channel covers the video addressable portion.

## New Flag Types

The `XrGlobalDimmerFrameEndInfoML::flags` member is of the following type, and contains a bitwise-OR of zero or more of the bits defined in `XrFrameEndInfoFlagBitsML`.

```
typedef XrFlags64 XrGlobalDimmerFrameEndInfoFlagsML;
```

Valid bits for `XrGlobalDimmerFrameEndInfoFlagsML` are defined by `XrGlobalDimmerFrameEndInfoFlagBitsML`, which is specified as:

```
// Flag bits for XrGlobalDimmerFrameEndInfoFlagsML
static const XrGlobalDimmerFrameEndInfoFlagsML
XR_GLOBAL_DIMMER_FRAME_END_INFO_ENABLED_BIT_ML = 0x00000001;
```

The flag bits have the following meanings:

### Flag Descriptions

- `XR_GLOBAL_DIMMER_FRAME_END_INFO_ENABLED_BIT_ML` — Indicates that the global dimmer **should** be enabled and controlled by `XrGlobalDimmerFrameEndInfoML::dimmerValue`.

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_GLOBAL_DIMMER_FRAME_END_INFO_ML`

## New Structures

The `XrGlobalDimmerFrameEndInfoML` structure is defined as:

```
// Provided by XR_ML_global_dimmer
typedef struct XrGlobalDimmerFrameEndInfoML {
    XrStructureType          type;
    const void*              next;
    float                    dimmerValue;
    XrGlobalDimmerFrameEndInfoFlagsML flags;
} XrGlobalDimmerFrameEndInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `dimmerValue` is a value between 0.0 (transparent) and 1.0 (opaque). The runtime **may** adjust the `dimmerValue` used during composition at the runtime’s discretion. This **may** be done for user safety, display performance, or other reasons. Values outside of the range are silently clamped.
- `flags` is a bitmask of [XrGlobalDimmerFrameEndInfoFlagsML](#)

## Valid Usage (Implicit)

- The [XR\\_ML\\_global\\_dimmer](#) extension **must** be enabled prior to using [XrGlobalDimmerFrameEndInfoML](#)
- `type` **must** be `XR_TYPE_GLOBAL_DIMMER_FRAME_END_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `flags` **must** be `0` or a valid combination of [XrGlobalDimmerFrameEndInfoFlagBitsML](#) values

## Version History

- Revision 1, 2022-10-25 (Ron Bessems)
  - Initial extension description

# 12.106. XR\_ML\_localization\_map

## Name String

`XR_ML_localization_map`

## Extension Type

Instance extension

## Registered Extension Number

140

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_EXT\\_uuid](#)

## Last Modified Date

2023-09-14

## Contributors

Ron Bessems, Magic Leap

Karthik Kadappan, Magic Leap

## 12.106.1. Overview

A Magic Leap localization map is a container that holds metadata about the scanned environment. It is a digital copy of a physical place. A localization map holds spatial anchors, dense mesh, planes, feature points, and positional data.

- Spatial anchors - Used for persistent placement of content.
- Dense mesh - 3D triangulated geometry representing Magic Leap device understanding of the real-world geometry of an area.
- Planes - Large, flat surfaces derived from dense mesh data.

Localization maps **can** be created on device or in the Magic Leap AR Cloud. There are two types - "On Device" and "Cloud".

- "On Device" for OpenXR (local space for MagicLeap) - are for a single device and **can** be shared via the export/import mechanism.
- "Cloud" for OpenXR (shared space for MagicLeap) - **can** be shared across multiple MagicLeap devices in the AR Cloud.



*Note*

Localization Maps are called Spaces in the Magic Leap C-API.

### Permissions

Android applications **must** have the `com.magicleap.permission.SPACE_MANAGER` permission listed in their manifest to use these functions:

- [xrQueryLocalizationMapsML](#)
- [xrRequestMapLocalizationML](#)

(protection level: normal)



Android applications **must** have the `com.magicleap.permission.SPACE_IMPORT_EXPORT` permission listed in their manifest and granted to use these functions:

- [xrImportLocalizationMapML](#)
- [xrCreateExportedLocalizationMapML](#)

(protection level: dangerous)

## 12.106.2. Current Localization Map Information

Applications **can** receive notifications when the current localization map changes by calling [xrPollEvent](#) and handling the [XrEventDataLocalizationChangedML](#) type. To enable these events call [xrEnableLocalizationEventsML](#).

The [XrEventDataLocalizationChangedML](#) structure is defined as:

```
// Provided by XR_ML_localization_map
typedef struct XrEventDataLocalizationChangedML {
    XrStructureType          type;
    const void*              next;
    XrSession                 session;
    XrLocalizationMapStateML state;
    XrLocalizationMapML       map;
    XrLocalizationMapConfidenceML confidence;
    XrLocalizationMapErrorFlagsML errorFlags;
} XrEventDataLocalizationChangedML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `session` is the session to which this change event applies.
- `state` is the current [XrLocalizationMapStateML](#) of the map.
- `map` is the [XrLocalizationMapML](#) of the current map.
- `confidence` is the [XrLocalizationMapConfidenceML](#) of the current map.
- `errorFlags` is a bitwise-OR of zero or more of the bits defined in [XrLocalizationMapErrorFlagBitsML](#) in the case that the localization map has low confidence.

By default the runtime does not send these events but calling [xrEnableLocalizationEventsML](#) function enables the events. When this function is called the [XrEventDataLocalizationChangedML](#) event will always be posted to the event queue, regardless of whether the map localization state has changed. This allows the application to synchronize with the current state.



### Note

The arrival of the event is asynchronous to this call.

## Valid Usage (Implicit)

- The [XR\\_ML\\_localization\\_map](#) extension **must** be enabled prior to using [XrEventDataLocalizationChangedML](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_LOCALIZATION_CHANGED_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The bitmask type [XrLocalizationMapErrorFlagsML](#) is defined as:

```
// Provided by XR_ML_localization_map
typedef XrFlags64 XrLocalizationMapErrorFlagsML;
```

As used in [XrEventDataLocalizationChangedML::errorFlags](#) field, [XrLocalizationMapErrorFlagsML](#) contains a bitwise-OR of zero or more of the bits defined in [XrLocalizationMapErrorFlagBitsML](#).

```

// Provided by XR_ML_localization_map
// Flag bits for XrLocalizationMapErrorFlagsML
static const XrLocalizationMapErrorFlagsML XR_LOCALIZATION_MAP_ERROR_UNKNOWN_BIT_ML =
0x00000001;
static const XrLocalizationMapErrorFlagsML
XR_LOCALIZATION_MAP_ERROR_OUT_OF_MAPPED_AREA_BIT_ML = 0x00000002;
static const XrLocalizationMapErrorFlagsML
XR_LOCALIZATION_MAP_ERROR_LOW_FEATURE_COUNT_BIT_ML = 0x00000004;
static const XrLocalizationMapErrorFlagsML
XR_LOCALIZATION_MAP_ERROR_EXCESSIVE_MOTION_BIT_ML = 0x00000008;
static const XrLocalizationMapErrorFlagsML XR_LOCALIZATION_MAP_ERROR_LOW_LIGHT_BIT_ML =
0x00000010;
static const XrLocalizationMapErrorFlagsML XR_LOCALIZATION_MAP_ERROR_HEADPOSE_BIT_ML =
0x00000020;

```

The flag bits have the following meanings:

### Flag Descriptions

- `XR_LOCALIZATION_MAP_ERROR_UNKNOWN_BIT_ML` — Localization failed for an unknown reason.
- `XR_LOCALIZATION_MAP_ERROR_OUT_OF_MAPPED_AREA_BIT_ML` — Localization failed because the user is outside of the mapped area.
- `XR_LOCALIZATION_MAP_ERROR_LOW_FEATURE_COUNT_BIT_ML` — There are not enough features in the environment to successfully localize.
- `XR_LOCALIZATION_MAP_ERROR_EXCESSIVE_MOTION_BIT_ML` — Localization failed due to excessive motion.
- `XR_LOCALIZATION_MAP_ERROR_LOW_LIGHT_BIT_ML` — Localization failed because the lighting levels are too low in the environment.
- `XR_LOCALIZATION_MAP_ERROR_HEADPOSE_BIT_ML` — A headpose failure caused localization to be unsuccessful.

The `xrEnableLocalizationEventsML` function is defined as:

```

// Provided by XR_ML_localization_map
XrResult xrEnableLocalizationEventsML(
    XrSession session,
    const XrLocalizationEnableEventsInfoML * info);

```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#) previously created with [xrCreateSession](#).
- `info` is a pointer to an [XrLocalizationEnableEventsInfoML](#) structure.

## Valid Usage (Implicit)

- The [XR\\_ML\\_localization\\_map](#) extension **must** be enabled prior to calling [xrEnableLocalizationEventsML](#)
- `session` **must** be a valid [XrSession](#) handle
- `info` **must** be a pointer to a valid [XrLocalizationEnableEventsInfoML](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_LOCALIZATION_MAP_PERMISSION_DENIED_ML`

The [XrLocalizationEnableEventsInfoML](#) structure is defined as:

```
// Provided by XR_ML_localization_map
typedef struct XrLocalizationEnableEventsInfoML {
    XrStructureType    type;
    const void*        next;
    XrBool32           enabled;
} XrLocalizationEnableEventsInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `enabled` is the flag to enable/disable localization status events.

## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to using [XrLocalizationEnableEventsInfoML](#)
- `type` **must** be `XR_TYPE_LOCALIZATION_ENABLE_EVENTS_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrLocalizationMapML](#) structure is defined as:

```
// Provided by XR_ML_localization_map
typedef struct XrLocalizationMapML {
    XrStructureType      type;
    void*                next;
    char                 name[XR_MAX_LOCALIZATION_MAP_NAME_LENGTH_ML];
    XrUuidEXT            mapUuid;
    XrLocalizationMapTypeML mapType;
} XrLocalizationMapML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `name` is a human readable name of the localization map, as a null terminated UTF-8 string. This name is set outside of this extension.
- `mapUuid` is the [XrUuidEXT](#) of the localization map.
- `mapType` is the [XrLocalizationMapTypeML](#) of the map.



## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to using `XrLocalizationMapML`
- `type` **must** be `XR_TYPE_LOCALIZATION_MAP_ML`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `name` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_LOCALIZATION_MAP_NAME_LENGTH_ML`
- If `mapType` is not `0`, `mapType` **must** be a valid `XrLocalizationMapTypeML` value

### 12.106.3. Listing Localization Maps

Localization maps available to the application **can** be queried using `xrQueryLocalizationMapsML`.

The `xrQueryLocalizationMapsML` function is defined as:

```
// Provided by XR_ML_localization_map
XrResult xrQueryLocalizationMapsML(
    XrSession session,
    const XrLocalizationMapQueryInfoBaseHeaderML* queryInfo,
    uint32_t mapCapacityInput,
    uint32_t* mapCountOutput,
    XrLocalizationMapML* maps);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession` previously created with `xrCreateSession`.
- `queryInfo` is an **optional** enumeration filter based on `XrLocalizationMapQueryInfoBaseHeaderML` to use.
- `mapCapacityInput` is the capacity of the maps array, or 0 to indicate a request to retrieve the required capacity.
- `mapCountOutput` is filled in by the runtime with the count of `maps` written or the required capacity in the case that `mapCapacityInput` is insufficient.
- `maps` is an array of `XrLocalizationMapML` filled in by the runtime, but **can** be `NULL` if `mapCapacityInput` is `0`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `maps` size.

The list of localization maps returned will depend on the current device mapping mode. Only the localization maps associated with the current mapping mode will be returned by this call. Device mapping mode (e.g. `XR_LOCALIZATION_MAP_TYPE_ON_DEVICE_ML` or `XR_LOCALIZATION_MAP_TYPE_CLOUD_ML`) **can** only be changed via the system application(s).

The list of maps known to the runtime **may** change between the two calls to `xrQueryLocalizationMapsML`. This is however a rare occurrence and the application **may** retry the call again if it receives `XR_ERROR_SIZE_INSUFFICIENT`.

### Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to calling `xrQueryLocalizationMapsML`
- `session` **must** be a valid `XrSession` handle
- If `queryInfo` is not `NULL`, `queryInfo` **must** be a pointer to a valid `XrLocalizationMapQueryInfoBaseHeaderML`-based structure
- `mapCountOutput` **must** be a pointer to a `uint32_t` value
- If `mapCapacityInput` is not `0`, `maps` **must** be a pointer to an array of `mapCapacityInput` `XrLocalizationMapML` structures

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_LOCALIZATION_MAP_PERMISSION_DENIED_ML`

The `XrLocalizationMapQueryInfoBaseHeaderML` structure is defined as:

```
// Provided by XR_ML_localization_map
typedef struct XrLocalizationMapQueryInfoBaseHeaderML {
    XrStructureType    type;
    const void*        next;
} XrLocalizationMapQueryInfoBaseHeaderML;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.

Currently no filters are available.

### Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to using [XrLocalizationMapQueryInfoBaseHeaderML](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## 12.106.4. Request Localization Map

Applications **can** change the current map by calling [xrRequestMapLocalizationML](#).

The [xrRequestMapLocalizationML](#) function is defined as:

```
// Provided by XR_ML_localization_map
XrResult xrRequestMapLocalizationML(
    XrSession session,
    const XrMapLocalizationRequestInfoML* requestInfo);
```

### Parameter Descriptions

- `session` is a handle to an [XrSession](#) previously created with [xrCreateSession](#).
- `requestInfo` contains [XrMapLocalizationRequestInfoML](#) on the localization map to request.

This is an asynchronous request. Listen for [XrEventDataLocalizationChangedML](#) events to get the results of the localization. A new request for localization will override all the past requests for

localization that are yet to be completed.

The runtime **must** return `XR_ERROR_LOCALIZATION_MAP_UNAVAILABLE_ML` if the requested is not a map known to the runtime.

### Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to calling `xrRequestMapLocalizationML`
- `session` **must** be a valid `XrSession` handle
- `requestInfo` **must** be a pointer to a valid `XrMapLocalizationRequestInfoML` structure

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_LOCALIZATION_MAP_UNAVAILABLE_ML`
- `XR_ERROR_LOCALIZATION_MAP_PERMISSION_DENIED_ML`
- `XR_ERROR_LOCALIZATION_MAP_FAIL_ML`

The `XrMapLocalizationRequestInfoML` structure is defined as:

```
// Provided by XR_ML_localization_map
typedef struct XrMapLocalizationRequestInfoML {
    XrStructureType    type;
    const void*        next;
    XrUuidEXT          mapUuid;
} XrMapLocalizationRequestInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `mapUuid` is the [XrUuidEXT](#) of the localization map to request. This `mapUuid` **can** be obtained via [xrQueryLocalizationMapsML](#).

## Valid Usage (Implicit)

- The [XR\\_ML\\_localization\\_map](#) extension **must** be enabled prior to using [XrMapLocalizationRequestInfoML](#)
- `type` **must** be `XR_TYPE_MAP_LOCALIZATION_REQUEST_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.106.5. Import and Exporting

This API supports exporting and importing of device localization maps. The runtime **must** not export AR Cloud maps and **must** return `XR_ERROR_LOCALIZATION_MAP_CANNOT_EXPORT_CLOUD_MAP_ML` if the application attempts to do so.

The format of the exported localization map data **can** change with OS version updates.

- Backwards compatibility: exports using OS version `n` **should** work on OS versions up to and including OS version `n-4`.
- Forwards compatibility: exports using OS version `n` is not guaranteed to work on OS versions `> n`.

Developers are strongly encouraged to encrypt the exported localization maps.

The [xrImportLocalizationMapML](#) function is defined as:

```
// Provided by XR_ML_localization_map
XrResult xrImportLocalizationMapML(
    XrSession session,
    const XrLocalizationMapImportInfoML* importInfo,
    XrUuidEXT* mapUuid);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#) previously created with [xrCreateSession](#).
- `importInfo` contains [XrLocalizationMapImportInfoML](#) on the localization map to import.
- `mapUuid` is the [XrUuidEXT](#) of the newly imported localization map filled in by the runtime.

The runtime **must** return `XR_ERROR_LOCALIZATION_MAP_ALREADY_EXISTS_ML` if the map that is being imported already exists. The runtime **must** return `XR_ERROR_LOCALIZATION_MAP_INCOMPATIBLE_ML` if the map being imported is not compatible.

[xrImportLocalizationMapML](#) **may** take a long time to complete; as such applications **should** not call this from the frame loop.

## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to calling [xrImportLocalizationMapML](#)
- `session` **must** be a valid [XrSession](#) handle
- `importInfo` **must** be a pointer to a valid [XrLocalizationMapImportInfoML](#) structure
- If `mapUuid` is not `NULL`, `mapUuid` **must** be a pointer to an [XrUuidEXT](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_LOCALIZATION_MAP_INCOMPATIBLE_ML`
- `XR_ERROR_LOCALIZATION_MAP_IMPORT_EXPORT_PERMISSION_DENIED_ML`
- `XR_ERROR_LOCALIZATION_MAP_ALREADY_EXISTS_ML`

The `XrLocalizationMapImportInfoML` structure is defined as:

```
// Provided by XR_ML_localization_map
typedef struct XrLocalizationMapImportInfoML {
    XrStructureType    type;
    const void*        next;
    uint32_t           size;
    char*              data;
} XrLocalizationMapImportInfoML;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `size` is the size in bytes of the data member.
- `data` is the byte data of the previously exported localization map.

## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to using `XrLocalizationMapImportInfoML`
- `type` **must** be `XR_TYPE_LOCALIZATION_MAP_IMPORT_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `data` **must** be a pointer to an array of `size` char values
- The `size` parameter **must** be greater than `0`

## Exporting

The `xrCreateExportedLocalizationMapML` function is defined as:

```
// Provided by XR_ML_localization_map
XrResult xrCreateExportedLocalizationMapML(
    XrSession                session,
    const XrUuidEXT*        mapUuid,
    XrExportedLocalizationMapML* map);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession` previously created with `xrCreateSession`.
- `mapUuid` is a pointer to the uuid of the map to export.
- `map` is a pointer to a map handle filled in by the runtime.

`xrCreateExportedLocalizationMapML` creates a frozen copy of the `mapUuid` localization map that **can** be exported using `xrGetExportedLocalizationMapDataML`. Applications **should** call `xrDestroyExportedLocalizationMapML` once they are done with the data.

## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to calling `xrCreateExportedLocalizationMapML`
- `session` **must** be a valid `XrSession` handle
- `mapUuid` **must** be a pointer to a valid `XrUuidEXT` structure
- `map` **must** be a pointer to an `XrExportedLocalizationMapML` handle



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_LOCALIZATION_MAP_UNAVAILABLE_ML`
- `XR_ERROR_LOCALIZATION_MAP_IMPORT_EXPORT_PERMISSION_DENIED_ML`
- `XR_ERROR_LOCALIZATION_MAP_CANNOT_EXPORT_CLOUD_MAP_ML`

The `xrDestroyExportedLocalizationMapML` function is defined as:

```
// Provided by XR_ML_localization_map
XrResult xrDestroyExportedLocalizationMapML(
    XrExportedLocalizationMapML    map);
```

## Parameter Descriptions

- `map` is the map to destroy.

## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to calling `xrDestroyExportedLocalizationMapML`
- `map` **must** be a valid `XrExportedLocalizationMapML` handle

## Thread Safety

- Access to `map`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`

The `xrGetExportedLocalizationMapDataML` function is defined as:

```
// Provided by XR_ML_localization_map
XrResult xrGetExportedLocalizationMapDataML(
    XrExportedLocalizationMapML          map,
    uint32_t                             bufferCapacityInput,
    uint32_t*                             bufferCountOutput,
    char*                                 buffer);
```

## Parameter Descriptions

- `map` is the map to export.
- `bufferCapacityInput` is the capacity of the buffer array, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is filled in by the runtime with the count of bytes written or the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is an array of bytes filled in by the runtime.

`xrGetExportedLocalizationMapDataML` **may** take a long time to complete; as such applications **should** not call this from the frame loop.

## Valid Usage (Implicit)

- The `XR_ML_localization_map` extension **must** be enabled prior to calling `xrGetExportedLocalizationMapDataML`
- `map` **must** be a valid `XrExportedLocalizationMapML` handle
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

## 12.106.6. Reference Space

Applications localized into the same localization map **can** use this reference space to place virtual content in the same physical location.

`XR_REFERENCE_SPACE_TYPE_LOCALIZATION_MAP_ML` is the reference space of the current localization map. Creating a space is done via `xrCreateReferenceSpace`.

The runtime **must** emit the `XrEventDataReferenceSpaceChangePending` event if the reference space is changing due to a localization map change.

The runtime **may** move the physical location of the origin of this space as it updates its understanding of the physical space to maintain consistency without sending the `XrEventDataReferenceSpaceChangePending` event.

For a given `XrUuidEXT` the runtime **must** keep the position and orientation of this space identical across more than one `XrInstance`, including for different users and different hardware.

The runtime **must** create this reference space as gravity-aligned to exclude pitch and roll, with +Y up.

### 12.106.7. Example code

The following code shows how to list the currently available localization maps.

```
uint32_t mapCount = 0;
CHK_XR(xrQueryLocalizationMapsML(session, nullptr, 0, &mapCount, nullptr));

std::vector<XrLocalizationMapML> maps(mapCount, {XR_TYPE_LOCALIZATION_MAP_ML});

CHK_XR(xrQueryLocalizationMapsML(session, nullptr, static_cast<uint32_t>(maps.size()),
&mapCount, maps.data()));
```

This code shows how to poll for localization events.

```
XrEventDataBuffer event{XR_TYPE_EVENT_DATA_BUFFER};
XrResult result = xrPollEvent(instance, &event);
if (result == XR_SUCCESS) {
    switch (event.type) {
        case XR_TYPE_EVENT_DATA_LOCALIZATION_CHANGED_ML: {
            const auto& localization_event =
                *reinterpret_cast<XrEventDataLocalizationChangedML*>(&event);
            // Use the data in localization_event.
            break;
        }
        // Handle other events as well as usual.
    }
}
```

### 12.106.8. Constants

#### New Object Types

```
XR_DEFINE_HANDLE(XrExportedLocalizationMapML)
```

[XrExportedLocalizationMapML](#) represents a frozen exported localization map.

#### New Enum Constants

[XrStructureType](#) enumeration is extended with:

- XR\_TYPE\_LOCALIZATION\_MAP\_ML
- XR\_TYPE\_EVENT\_DATA\_LOCALIZATION\_CHANGED\_ML
- XR\_TYPE\_MAP\_LOCALIZATION\_REQUEST\_INFO\_ML
- XR\_TYPE\_LOCALIZATION\_MAP\_IMPORT\_INFO\_ML
- XR\_TYPE\_LOCALIZATION\_ENABLE\_EVENTS\_INFO\_ML

**XrResult** enumeration is extended with:

- XR\_ERROR\_LOCALIZATION\_MAP\_INCOMPATIBLE\_ML
- XR\_ERROR\_LOCALIZATION\_MAP\_UNAVAILABLE\_ML
- XR\_ERROR\_LOCALIZATION\_MAP\_IMPORT\_EXPORT\_PERMISSION\_DENIED\_ML
- XR\_ERROR\_LOCALIZATION\_MAP\_PERMISSION\_DENIED\_ML
- XR\_ERROR\_LOCALIZATION\_MAP\_ALREADY\_EXISTS\_ML
- XR\_ERROR\_LOCALIZATION\_MAP\_CANNOT\_EXPORT\_CLOUD\_MAP\_ML
- XR\_ERROR\_LOCALIZATION\_MAP\_FAIL\_ML

## New Enums

```
// Provided by XR_ML_localization_map
typedef enum XrLocalizationMapStateML {
    XR_LOCALIZATION_MAP_STATE_NOT_LOCALIZED_ML = 0,
    XR_LOCALIZATION_MAP_STATE_LOCALIZED_ML = 1,
    XR_LOCALIZATION_MAP_STATE_LOCALIZATION_PENDING_ML = 2,
    XR_LOCALIZATION_MAP_STATE_LOCALIZATION_SLEEPING_BEFORE_RETRY_ML = 3,
    XR_LOCALIZATION_MAP_STATE_MAX_ENUM_ML = 0x7FFFFFFF
} XrLocalizationMapStateML;
```

| Enum  | Description  |
|---|--|
| XR_LOCALIZATION_MAP_STATE_NOT_LOCALIZED_ML                      | The system is not localized into a map. Features like Spatial Anchors relying on localization will not work. |
| XR_LOCALIZATION_MAP_STATE_LOCALIZED_ML                          | The system is localized into a map.  |
| XR_LOCALIZATION_MAP_STATE_LOCALIZATION_PENDING_ML               | The system is localizing into a map.   |
| XR_LOCALIZATION_MAP_STATE_LOCALIZATION_SLEEPING_BEFORE_RETRY_ML | Initial localization failed, the system will retry localization.   |

```
// Provided by XR_ML_localization_map
typedef enum XrLocalizationMapConfidenceML {
    XR_LOCALIZATION_MAP_CONFIDENCE_POOR_ML = 0,
    XR_LOCALIZATION_MAP_CONFIDENCE_FAIR_ML = 1,
    XR_LOCALIZATION_MAP_CONFIDENCE_GOOD_ML = 2,
    XR_LOCALIZATION_MAP_CONFIDENCE_EXCELLENT_ML = 3,
    XR_LOCALIZATION_MAP_CONFIDENCE_MAX_ENUM_ML = 0x7FFFFFFF
} XrLocalizationMapConfidenceML;
```

| Enum   | Description  |
|--|--|
| <code>XR_LOCALIZATION_MAP_CONFIDENCE_POOR_ML</code>      | The localization map has poor confidence, systems relying on the localization map are likely to have poor performance. |
| <code>XR_LOCALIZATION_MAP_CONFIDENCE_FAIR_ML</code>      | The confidence is fair, current environmental conditions may adversely affect localization.                            |
| <code>XR_LOCALIZATION_MAP_CONFIDENCE_GOOD_ML</code>      | The confidence is high, persistent content should be stable.   |
| <code>XR_LOCALIZATION_MAP_CONFIDENCE_EXCELLENT_ML</code> | This is a very high-confidence localization, persistent content will be very stable.                                   |

```
// Provided by XR_ML_localization_map
typedef enum XrLocalizationMapTypeML {
    XR_LOCALIZATION_MAP_TYPE_ON_DEVICE_ML = 0,
    XR_LOCALIZATION_MAP_TYPE_CLOUD_ML = 1,
    XR_LOCALIZATION_MAP_TYPE_MAX_ENUM_ML = 0x7FFFFFFF
} XrLocalizationMapTypeML;
```

| Enum   | Description  |
|--|--|
| <code>XR_LOCALIZATION_MAP_TYPE_ON_DEVICE_ML</code> | The system is localized into an On-Device map, published anchors are not shared between different devices. |
| <code>XR_LOCALIZATION_MAP_TYPE_CLOUD_ML</code>     | The system is localized into a Cloud Map, anchors are shared per cloud account settings.                   |

## New Enum Constants

`XrReferenceSpaceType` enumeration is extended with:

- `XR_REFERENCE_SPACE_TYPE_LOCALIZATION_MAP_ML`

## New Defines

## Version History

- Revision 1, 2023-06-23 (Ron Bessems)
  - Initial extension description

# 12.107. XR\_ML\_marker\_understanding

## Name String

`XR_ML_marker_understanding`

## Extension Type

Instance extension

## Registered Extension Number

139

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2023-05-18

## Contributors

Robbie Bridgewater, Magic Leap  
Ron Bessems, Magic Leap  
Karthik Kadappan, Magic Leap

## 12.107.1. Overview

This extension **can** be used to track and query fiducial markers like QR codes, AprilTag markers, and ArUco markers, and detect, but not locate, 1D barcodes like Code 128, UPC-A.



### *Permissions*

Android applications **must** have the `com.magicleap.permission.MARKER_TRACKING` permission listed in their manifest to use this extension. (protection level: normal)

## 12.107.2. Creating a Marker Detector

```
// Provided by XR_ML_marker_understanding
XR_DEFINE_HANDLE(XrMarkerDetectorML)
```

The [XrMarkerDetectorML](#) handle represents the resources for detecting one or more markers.

A marker detector handle detects a single type of marker, specified by a value of [XrMarkerTypeML](#). To detect more than one marker type, a runtime **may** support creating multiple marker detector handles.

This handle **can** be used to detect markers using other functions in this extension.

The [xrCreateMarkerDetectorML](#) function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrCreateMarkerDetectorML(
    XrSession session,
    const XrMarkerDetectorCreateInfoML* createInfo,
    XrMarkerDetectorML* markerDetector);
```

## Parameter Descriptions

- `session` is an [XrSession](#) in which the marker detection will be active.
- `createInfo` is the [XrMarkerDetectorCreateInfoML](#) used to specify the marker detection.
- `markerDetector` is the returned [XrMarkerDetectorML](#) handle.

An application creates an [XrMarkerDetectorML](#) handle using the [xrCreateMarkerDetectorML](#) function. If `createInfo` contains mutually exclusive contents, the runtime **must** return [XR\\_ERROR\\_MARKER\\_DETECTOR\\_INVALID\\_CREATE\\_INFO\\_ML](#).

If a runtime is unable to create a marker detector due to some internal limit, the runtime **must** return [XR\\_ERROR\\_LIMIT\\_REACHED](#).

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to calling [xrCreateMarkerDetectorML](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrMarkerDetectorCreateInfoML](#) structure
- `markerDetector` **must** be a pointer to an [XrMarkerDetectorML](#) handle



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_MARKER_DETECTOR_PERMISSION_DENIED_ML`
- `XR_ERROR_MARKER_DETECTOR_INVALID_CREATE_INFO_ML`

The `XrMarkerDetectorCreateInfoML` structure is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorCreateInfoML {
    XrStructureType          type;
    const void*              next;
    XrMarkerDetectorProfileML profile;
    XrMarkerTypeML           markerType;
} XrMarkerDetectorCreateInfoML;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `profile` is the marker tracker profile to be used.
- `markerType` is the detector type that this tracker enables.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to using `XrMarkerDetectorCreateInfoML`
- `type` **must** be `XR_TYPE_MARKER_DETECTOR_CREATE_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the next structure in a structure chain. See also: `XrMarkerDetectorAprilTagInfoML`, `XrMarkerDetectorArucoInfoML`, `XrMarkerDetectorCustomProfileInfoML`, `XrMarkerDetectorSizeInfoML`
- `profile` **must** be a valid `XrMarkerDetectorProfileML` value
- `markerType` **must** be a valid `XrMarkerTypeML` value

The possible premade profiles for an `XrMarkerDetectorML` are specified by the `XrMarkerDetectorProfileML` enumeration:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorProfileML {
    XR_MARKER_DETECTOR_PROFILE_DEFAULT_ML = 0,
    XR_MARKER_DETECTOR_PROFILE_SPEED_ML = 1,
    XR_MARKER_DETECTOR_PROFILE_ACCURACY_ML = 2,
    XR_MARKER_DETECTOR_PROFILE_SMALL_TARGETS_ML = 3,
    XR_MARKER_DETECTOR_PROFILE_LARGE_FOV_ML = 4,
    XR_MARKER_DETECTOR_PROFILE_CUSTOM_ML = 5,
    XR_MARKER_DETECTOR_PROFILE_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorProfileML;
```

## Enumerant Descriptions

- `XR_MARKER_DETECTOR_PROFILE_DEFAULT_ML` — Tracker profile that covers standard use cases. If this does not suite the needs of the application try the other profiles listed below.
- `XR_MARKER_DETECTOR_PROFILE_SPEED_ML` — Optimized for speed. Use this profile to reduce the compute load and increase detection/tracker speed. This can result in low accuracy poses.
- `XR_MARKER_DETECTOR_PROFILE_ACCURACY_ML` — Optimized for accuracy. Use this profile to optimize for accurate marker poses. This can cause increased load on the compute.
- `XR_MARKER_DETECTOR_PROFILE_SMALL_TARGETS_ML` — Optimized for small targets. Use this profile to optimize for markers that are small or for larger markers that need to be detected from afar.
- `XR_MARKER_DETECTOR_PROFILE_LARGE_FOV_ML` — Optimized for FoV. Use this profile to be able to detect markers across a larger FoV. The marker tracker system will attempt to use multiple cameras to detect the markers.
- `XR_MARKER_DETECTOR_PROFILE_CUSTOM_ML` — Custom Tracker Profile. The application can define a custom tracker profile. See [XrMarkerDetectorCustomProfileInfoML](#) for more details.

The type of marker to be tracked is specified via [XrMarkerDetectorML](#):

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerTypeML {
    XR_MARKER_TYPE_ARUCO_ML = 0,
    XR_MARKER_TYPE_APRIL_TAG_ML = 1,
    XR_MARKER_TYPE_QR_ML = 2,
    XR_MARKER_TYPE_EAN_13_ML = 3,
    XR_MARKER_TYPE_UPC_A_ML = 4,
    XR_MARKER_TYPE_CODE_128_ML = 5,
    XR_MARKER_TYPE_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerTypeML;
```

## Enumerant Descriptions

- `XR_MARKER_TYPE_ARUCO_ML` — Aruco Marker detection and localization. The marker id of the Aruco marker is available via [xrGetMarkerNumberML](#).
- `XR_MARKER_TYPE_APRIL_TAG_ML` — AprilTag detection and localization. The marker id of the AprilTags is available via [xrGetMarkerNumberML](#).
- `XR_MARKER_TYPE_QR_ML` — QR code detection and localization. The contents of the QR code is available via [xrGetMarkerStringML](#).
- `XR_MARKER_TYPE_EAN_13_ML` — EAN-13, detection only, not locatable. The contents of the barcode is available via [xrGetMarkerStringML](#).
- `XR_MARKER_TYPE_UPC_A_ML` — UPC-A, detection only, not locatable. The contents of the barcode is available via [xrGetMarkerStringML](#).
- `XR_MARKER_TYPE_CODE_128_ML` — Code 128, detection only, not locatable. The contents of the barcode is available via [xrGetMarkerStringML](#).

An application specifies details of the type of marker to be tracked by chaining an `XrMarkerDetector*InfoML` structure to [XrMarkerDetectorCreateInfoML](#). Some of these structure types **must** be included to enable detection or locating, depending on the marker type.

The following structures are used by the ArUco, AprilTag, and QR code detectors:

| Marker Type | Structures   |
|-------------|--|
| ArUco       | <a href="#">XrMarkerDetectorArucoInfoML</a><br><a href="#">XrMarkerDetectorSizeInfoML</a>    |
| AprilTag    | <a href="#">XrMarkerDetectorAprilTagInfoML</a><br><a href="#">XrMarkerDetectorSizeInfoML</a> |
| QR Code     | <a href="#">XrMarkerDetectorSizeInfoML</a>   |

The [XrMarkerDetectorSizeInfoML](#) **may** be optional depending on runtime support for estimating marker size. A higher localization accuracy **may** be obtained by specifying the marker size. If the runtime does not support estimating marker size it **must** return `XR_ERROR_VALIDATION_FAILURE` if [XrMarkerDetectorSizeInfoML](#) is omitted.

The [XrMarkerDetectorArucoInfoML](#) structure extends [XrMarkerDetectorCreateInfoML](#) and is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorArucoInfoML {
    XrStructureType      type;
    const void*          next;
    XrMarkerArucoDictML  arucoDict;
} XrMarkerDetectorArucoInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `arucoDict` is the ArUco dictionary name from which markers will be detected.

This structure is required by the `XR_MARKER_TYPE_ARUCO_ML` detector.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to using [XrMarkerDetectorArucoInfoML](#)
- `type` **must** be `XR_TYPE_MARKER_DETECTOR_ARUCO_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `arucoDict` **must** be a valid [XrMarkerArucoDictML](#) value

The [XrMarkerArucoDictML](#) enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerArucoDictML {
    XR_MARKER_ARUCO_DICT_4X4_50_ML = 0,
    XR_MARKER_ARUCO_DICT_4X4_100_ML = 1,
    XR_MARKER_ARUCO_DICT_4X4_250_ML = 2,
    XR_MARKER_ARUCO_DICT_4X4_1000_ML = 3,
    XR_MARKER_ARUCO_DICT_5X5_50_ML = 4,
    XR_MARKER_ARUCO_DICT_5X5_100_ML = 5,
    XR_MARKER_ARUCO_DICT_5X5_250_ML = 6,
    XR_MARKER_ARUCO_DICT_5X5_1000_ML = 7,
    XR_MARKER_ARUCO_DICT_6X6_50_ML = 8,
    XR_MARKER_ARUCO_DICT_6X6_100_ML = 9,
    XR_MARKER_ARUCO_DICT_6X6_250_ML = 10,
    XR_MARKER_ARUCO_DICT_6X6_1000_ML = 11,
    XR_MARKER_ARUCO_DICT_7X7_50_ML = 12,
    XR_MARKER_ARUCO_DICT_7X7_100_ML = 13,
    XR_MARKER_ARUCO_DICT_7X7_250_ML = 14,
    XR_MARKER_ARUCO_DICT_7X7_1000_ML = 15,
    XR_MARKER_ARUCO_DICT_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerArucoDictML;
```

Supported predefined ArUco dictionary:

## Enumerant Descriptions

- `XR_MARKER_ARUCO_DICT_4X4_50_ML` — 4 by 4 pixel Aruco marker dictionary with 50 IDs.
- `XR_MARKER_ARUCO_DICT_4X4_100_ML` — 4 by 4 pixel Aruco marker dictionary with 100 IDs.
- `XR_MARKER_ARUCO_DICT_4X4_250_ML` — 4 by 4 pixel Aruco marker dictionary with 250 IDs.
- `XR_MARKER_ARUCO_DICT_4X4_1000_ML` — 4 by 4 pixel Aruco marker dictionary with 1000 IDs.
- `XR_MARKER_ARUCO_DICT_5X5_50_ML` — 5 by 5 pixel Aruco marker dictionary with 50 IDs.
- `XR_MARKER_ARUCO_DICT_5X5_100_ML` — 5 by 5 pixel Aruco marker dictionary with 100 IDs.
- `XR_MARKER_ARUCO_DICT_5X5_250_ML` — 5 by 5 pixel Aruco marker dictionary with 250 IDs.
- `XR_MARKER_ARUCO_DICT_5X5_1000_ML` — 5 by 5 pixel Aruco marker dictionary with 1000 IDs.
- `XR_MARKER_ARUCO_DICT_6X6_50_ML` — 6 by 6 pixel Aruco marker dictionary with 50 IDs.
- `XR_MARKER_ARUCO_DICT_6X6_100_ML` — 6 by 6 pixel Aruco marker dictionary with 100 IDs.
- `XR_MARKER_ARUCO_DICT_6X6_250_ML` — 6 by 6 pixel Aruco marker dictionary with 250 IDs.
- `XR_MARKER_ARUCO_DICT_6X6_1000_ML` — 6 by 6 pixel Aruco marker dictionary with 1000 IDs.
- `XR_MARKER_ARUCO_DICT_7X7_50_ML` — 7 by 7 pixel Aruco marker dictionary with 50 IDs.
- `XR_MARKER_ARUCO_DICT_7X7_100_ML` — 7 by 7 pixel Aruco marker dictionary with 100 IDs.
- `XR_MARKER_ARUCO_DICT_7X7_250_ML` — 7 by 7 pixel Aruco marker dictionary with 250 IDs.
- `XR_MARKER_ARUCO_DICT_7X7_1000_ML` — 7 by 7 pixel Aruco marker dictionary with 1000 IDs.

The `XrMarkerDetectorAprilTagInfoML` structure extends `XrMarkerDetectorCreateInfoML` and is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorAprilTagInfoML {
    XrStructureType          type;
    const void*              next;
    XrMarkerAprilTagDictML   aprilTagDict;
} XrMarkerDetectorAprilTagInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `aprilTagDict` AprilTag Dictionary name from which markers will be detected.

This structure is required by the `XR_MARKER_TYPE_APRIL_TAG_ML` detector.

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to using [XrMarkerDetectorAprilTagInfoML](#)
- `type` **must** be `XR_TYPE_MARKER_DETECTOR_APRIL_TAG_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `aprilTagDict` **must** be a valid [XrMarkerAprilTagDictML](#) value

The [XrMarkerAprilTagDictML](#) enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerAprilTagDictML {
    XR_MARKER_APRIL_TAG_DICT_16H5_ML = 0,
    XR_MARKER_APRIL_TAG_DICT_25H9_ML = 1,
    XR_MARKER_APRIL_TAG_DICT_36H10_ML = 2,
    XR_MARKER_APRIL_TAG_DICT_36H11_ML = 3,
    XR_MARKER_APRIL_TAG_DICT_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerAprilTagDictML;
```

Supported predefined AprilTag dictionary:



## Enumerant Descriptions

- `XR_MARKER_APRIIL_TAG_DICT_16H5_ML` — 4 by 4 bits, minimum Hamming distance between any two codes = 5, 30 codes.
- `XR_MARKER_APRIIL_TAG_DICT_25H9_ML` — 5 by 5 bits, minimum Hamming distance between any two codes = 9, 35 codes.
- `XR_MARKER_APRIIL_TAG_DICT_36H10_ML` — 6 by 6 bits, minimum Hamming distance between any two codes = 10, 2320 codes.
- `XR_MARKER_APRIIL_TAG_DICT_36H11_ML` — 6 by 6 bits, minimum Hamming distance between any two codes = 11, 587 codes.

The `XrMarkerDetectorSizeInfoML` structure extends `XrMarkerDetectorCreateInfoML` and is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorSizeInfoML {
    XrStructureType    type;
    const void*        next;
    float              markerLength;
} XrMarkerDetectorSizeInfoML;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `markerLength` is the physical length of one side of a marker.

Pose estimation accuracy depends on the accuracy of the specified `markerLength`.

This structure is used by `XR_MARKER_TYPE_ARUCO_ML`, `XR_MARKER_TYPE_APRIIL_TAG_ML`, and `XR_MARKER_TYPE_QR_ML` detectors.

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to using [XrMarkerDetectorSizeInfoML](#)
- **type** **must** be `XR_TYPE_MARKER_DETECTOR_SIZE_INFO_ML`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [xrDestroyMarkerDetectorML](#) function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrDestroyMarkerDetectorML(
    XrMarkerDetectorML          markerDetector);
```

## Parameter Descriptions

- `markerDetector` object to destroy.

Destroy a marker detection handle.

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to calling [xrDestroyMarkerDetectorML](#)
- `markerDetector` **must** be a valid [XrMarkerDetectorML](#) handle

## Thread Safety

- Access to `markerDetector`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## Using a custom profile

The `XrMarkerDetectorCustomProfileInfoML` structure extends `XrMarkerDetectorCreateInfoML` and is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorCustomProfileInfoML {
    XrStructureType                type;
    const void*                    next;
    XrMarkerDetectorFpsML          fpsHint;
    XrMarkerDetectorResolutionML   resolutionHint;
    XrMarkerDetectorCameraML       cameraHint;
    XrMarkerDetectorCornerRefineMethodML cornerRefineMethod;
    XrBool32                       useEdgeRefinement;
    XrMarkerDetectorFullAnalysisIntervalML fullAnalysisIntervalHint;
} XrMarkerDetectorCustomProfileInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `fpsHint` is a suggestion of the category of frame rate for the detector to use.
- `resolutionHint` is a suggestion of the category of camera resolution for the detector to use.
- `cameraHint` is a suggestion of the camera set for the detector to use
- `cornerRefineMethod` selects a method for corner refinement for ArUco/AprilTag detectors. This member is ignored for detectors of other marker types.
- `useEdgeRefinement` specifies whether to run a refinement step that uses marker edges to generate even more accurate corners, but slow down tracking rate overall by consuming more compute. It affects ArUco/AprilTag markers only: this member is ignored for detectors of other marker types.
- `fullAnalysisIntervalHint` is the suggested interval between fully analyzed frames that introduce new detected markers, in addition to updating the state of already detected markers.

All marker detectors share some underlying hardware and resources, and thus not all combinations of profiles between multiple detectors are possible. If a profile (preset or custom) specified during marker detector creation is different from those used by existing marker detectors the runtime will attempt to honor the highest frame rate and fps requested.

CPU load due to marker tracking is a function of the chosen [XrMarkerTypeML](#), [XrMarkerDetectorFpsML](#), and [XrMarkerDetectorResolutionML](#).

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to using [XrMarkerDetectorCustomProfileInfoML](#)
- `type` **must** be `XR_TYPE_MARKER_DETECTOR_CUSTOM_PROFILE_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `fpsHint` **must** be a valid [XrMarkerDetectorFpsML](#) value
- `resolutionHint` **must** be a valid [XrMarkerDetectorResolutionML](#) value
- `cameraHint` **must** be a valid [XrMarkerDetectorCameraML](#) value
- `cornerRefineMethod` **must** be a valid [XrMarkerDetectorCornerRefineMethodML](#) value
- `fullAnalysisIntervalHint` **must** be a valid [XrMarkerDetectorFullAnalysisIntervalML](#) value

The [XrMarkerDetectorFpsML](#) enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorFpsML {
    XR_MARKER_DETECTOR_FPS_LOW_ML = 0,
    XR_MARKER_DETECTOR_FPS_MEDIUM_ML = 1,
    XR_MARKER_DETECTOR_FPS_HIGH_ML = 2,
    XR_MARKER_DETECTOR_FPS_MAX_ML = 3,
    XR_MARKER_DETECTOR_FPS_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorFpsML;
```

Used to hint to the back-end the max frames per second that **should** be analyzed.

### Enumerant Descriptions

- [XR\\_MARKER\\_DETECTOR\\_FPS\\_LOW\\_ML](#) — Low FPS.
- [XR\\_MARKER\\_DETECTOR\\_FPS\\_MEDIUM\\_ML](#) — Medium FPS.
- [XR\\_MARKER\\_DETECTOR\\_FPS\\_HIGH\\_ML](#) — High FPS.
- [XR\\_MARKER\\_DETECTOR\\_FPS\\_MAX\\_ML](#) — Max possible FPS.

The [XrMarkerDetectorResolutionML](#) enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorResolutionML {
    XR_MARKER_DETECTOR_RESOLUTION_LOW_ML = 0,
    XR_MARKER_DETECTOR_RESOLUTION_MEDIUM_ML = 1,
    XR_MARKER_DETECTOR_RESOLUTION_HIGH_ML = 2,
    XR_MARKER_DETECTOR_RESOLUTION_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorResolutionML;
```

Used to hint to the back-end the resolution that **should** be used. CPU load is a combination of chosen [XrMarkerTypeML](#), [XrMarkerDetectorFpsML](#), and [XrMarkerDetectorResolutionML](#).

## Enumerant Descriptions

- `XR_MARKER_DETECTOR_RESOLUTION_LOW_ML` — Low Resolution.
- `XR_MARKER_DETECTOR_RESOLUTION_MEDIUM_ML` — Medium Resolution.
- `XR_MARKER_DETECTOR_RESOLUTION_HIGH_ML` — High Resolution.

The `XrMarkerDetectorCameraML` enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorCameraML {
    XR_MARKER_DETECTOR_CAMERA_RGB_CAMERA_ML = 0,
    XR_MARKER_DETECTOR_CAMERA_WORLD_CAMERAS_ML = 1,
    XR_MARKER_DETECTOR_CAMERA_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorCameraML;
```

The `XrMarkerDetectorCameraML` enum values are used to hint to the camera that **should** be used. This is set in the `XrMarkerDetectorCustomProfileInfoML`.

The RGB camera has a higher resolution than world cameras and is better suited for use cases where the target to be tracked is small or needs to be detected from far away.

`XR_MARKER_DETECTOR_CAMERA_WORLD_CAMERAS_ML` make use of multiple cameras to improve accuracy and increase the FoV for detection.

## Enumerant Descriptions

- `XR_MARKER_DETECTOR_CAMERA_RGB_CAMERA_ML` — Single RGB camera.
- `XR_MARKER_DETECTOR_CAMERA_WORLD_CAMERAS_ML` — One or more world cameras.

The `XrMarkerDetectorCornerRefineMethodML` enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorCornerRefineMethodML {
    XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_NONE_ML = 0,
    XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_SUBPIX_ML = 1,
    XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_CONTOUR_ML = 2,
    XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_APRIL_TAG_ML = 3,
    XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorCornerRefineMethodML;
```

The ArUco/AprilTag detector comes with several corner refinement methods. Choosing the right corner refinement method has an impact on the accuracy and speed trade-off that comes with each detection pipeline.

### Enumerant Descriptions

- `XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_NONE_ML` — No refinement. Inaccurate corners.
- `XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_SUBPIX_ML` — Subpixel refinement. Corners have subpixel coordinates. High detection rate, very fast, reasonable accuracy.
- `XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_CONTOUR_ML` — Contour refinement. High detection rate, fast, reasonable accuracy.
- `XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_APRIL_TAG_ML` — AprilTag refinement. Reasonable detection rate, slowest, but very accurate. Only valid with AprilTags.

The `XrMarkerDetectorFullAnalysisIntervalML` enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorFullAnalysisIntervalML {
    XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_MAX_ML = 0,
    XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_FAST_ML = 1,
    XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_MEDIUM_ML = 2,
    XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_SLOW_ML = 3,
    XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorFullAnalysisIntervalML;
```

In order to improve performance, the detectors do not always run on the full frame. Full frame analysis is however necessary to detect new markers that were not detected before. Use this option to control how often the detector **should** detect new markers and its impact on tracking performance.

## Enumerant Descriptions

- `XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_MAX_ML` — Detector analyzes every frame fully.
- `XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_FAST_ML` — Detector analyzes frame fully very often.
- `XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_MEDIUM_ML` — Detector analyzes frame fully a few times per second.
- `XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_SLOW_ML` — Detector analyzes frame fully about every second.

### 12.107.3. Scanning for markers

The `xrSnapshotMarkerDetectorML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrSnapshotMarkerDetectorML(
    XrMarkerDetectorML          markerDetector,
    XrMarkerDetectorSnapshotInfoML* snapshotInfo);
```

## Parameter Descriptions

- `markerDetector` object to issue a snapshot request to.
- `snapshotInfo` is a pointer to `XrMarkerDetectorSnapshotInfoML` containing marker snapshot parameters.

Collects the latest marker detector state and makes it ready for inspection. This function only snapshots the non-pose state of markers. Once called, and if a new snapshot is not yet available a runtime **must** set the state of the marker detector to `XR_MARKER_DETECTOR_STATUS_PENDING_ML`. If a new state is available the runtime **must** set the state to `XR_MARKER_DETECTOR_STATUS_READY_ML`. If an error occurred the runtime **must** set the state to `XR_MARKER_DETECTOR_STATUS_ERROR_ML`. The application **may** attempt the snapshot again.

Once the application has inspected the state it is interested in it **can** call this function again and the state is set to `XR_MARKER_DETECTOR_STATUS_PENDING_ML` until a new state has been snapshotted. After each snapshot, only the currently detected markers are available for inspection, though the same marker **may** repeatedly be detected across snapshots.



## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrSnapshotMarkerDetectorML`
- `markerDetector` **must** be a valid `XrMarkerDetectorML` handle
- `snapshotInfo` **must** be a pointer to an `XrMarkerDetectorSnapshotInfoML` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

The `XrMarkerDetectorSnapshotInfoML` structure is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorSnapshotInfoML {
    XrStructureType    type;
    const void*        next;
} XrMarkerDetectorSnapshotInfoML;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to using `XrMarkerDetectorSnapshotInfoML`
- `type` **must** be `XR_TYPE_MARKER_DETECTOR_SNAPSHOT_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

The `xrGetMarkerDetectorStateML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrGetMarkerDetectorStateML(
    XrMarkerDetectorML          markerDetector,
    XrMarkerDetectorStateML*   state);
```

## Parameter Descriptions

- `markerDetector` object to retrieve state information from.
- `state` points to an `XrMarkerDetectorStateML` in which the current state of the marker detector is returned.

`xrGetMarkerDetectorStateML` is used after calling `xrSnapshotMarkerDetectorML` to check the current status of the snapshot in progress. When `XrMarkerDetectorStateML::state == XR_MARKER_DETECTOR_STATUS_READY_ML`, the detector is ready to be queried, while `XR_MARKER_DETECTOR_STATUS_PENDING_ML` indicates the snapshot is still in progress. `XR_MARKER_DETECTOR_STATUS_ERROR_ML` indicates that the runtime has encountered an error getting a snapshot for the requested detector, which **may** require user intervention to solve.

If `xrSnapshotMarkerDetectorML` has not yet been called for the `markerDetector`, the runtime **must** return `XR_ERROR_CALL_ORDER_INVALID`.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrGetMarkerDetectorStateML`
- `markerDetector` **must** be a valid `XrMarkerDetectorML` handle
- `state` **must** be a pointer to an `XrMarkerDetectorStateML` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_CALL_ORDER_INVALID`

The `XrMarkerDetectorStateML` structure is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerDetectorStateML {
    XrStructureType      type;
    void*                next;
    XrMarkerDetectorStatusML state;
} XrMarkerDetectorStateML;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `state` is the current state of the marker detector.

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to using [XrMarkerDetectorStateML](#)
- **type** **must** be [XR\\_TYPE\\_MARKER\\_DETECTOR\\_STATE\\_ML](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

The [XrMarkerDetectorStatusML](#) enumeration is defined as:

```
// Provided by XR_ML_marker_understanding
typedef enum XrMarkerDetectorStatusML {
    XR_MARKER_DETECTOR_STATUS_PENDING_ML = 0,
    XR_MARKER_DETECTOR_STATUS_READY_ML = 1,
    XR_MARKER_DETECTOR_STATUS_ERROR_ML = 2,
    XR_MARKER_DETECTOR_STATUS_MAX_ENUM_ML = 0x7FFFFFFF
} XrMarkerDetectorStatusML;
```

The [XrMarkerDetectorStatusML](#) enumeration describes the current state of the marker detector. It is queried via [xrGetMarkerDetectorStateML](#) to determine if the marker tracker is currently available for inspection.

## Enumerant Descriptions

- [XR\\_MARKER\\_DETECTOR\\_STATUS\\_PENDING\\_ML](#) — The marker detector is working on a new snapshot.
- [XR\\_MARKER\\_DETECTOR\\_STATUS\\_READY\\_ML](#) — The marker detector is ready to be inspected.
- [XR\\_MARKER\\_DETECTOR\\_STATUS\\_ERROR\\_ML](#) — The marker detector has encountered a fatal error.

### 12.107.4. Getting Marker Results

The [xrGetMarkersML](#) function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrGetMarkersML(
    XrMarkerDetectorML          markerDetector,
    uint32_t                    markerCapacityInput,
    uint32_t*                   markerCountOutput,
    XrMarkerML*                 markers);
```

## Parameter Descriptions

- `markerDetector` is the detector object to retrieve marker information from.
- `markerCapacityInput` is the capacity of the `markers` array or `0` to indicate a request to retrieve the required capacity.
- `markerCountOutput` is filled in by the runtime with the count of marker atoms written or the required capacity in the case that `markerCapacityInput` is insufficient.
- `markers` is a pointer to an array of `XrMarkerML` atoms, but **can** be `NULL` if `propertyCapacityInput` is `0`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `markers` size.

Get the list of current snapshotted marker atoms, **must** only be called when the state of the detector is `XR_MARKER_DETECTOR_STATUS_READY_ML`.

If `xrGetMarkerDetectorStateML` has not been called and returned `XR_MARKER_DETECTOR_STATUS_READY_ML` since the last invocation of `xrSnapshotMarkerDetectorML`, the runtime **must** return `XR_ERROR_CALL_ORDER_INVALID`.

The returned atoms are only valid while in the `XR_MARKER_DETECTOR_STATUS_READY_ML` state. The runtime **must** return the same atom value for the same uniquely identifiable marker across successive snapshots. It is unspecified what happens if the detector is observing two markers with the same identification patterns.

Assuming the same set of markers are in view across several snapshots, the runtime **should** return the same set of atoms. An application **can** use the list of atoms as a simple test for if a particular marker has gone in or out of view.

Note that `XrMarkerML` atoms are only usable with the `XrMarkerDetectorML` that returned them.

This function follows the [two-call idiom](#) for filling the `markers`.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrGetMarkersML`
- `markerDetector` **must** be a valid `XrMarkerDetectorML` handle
- `markerCountOutput` **must** be a pointer to a `uint32_t` value
- If `markerCapacityInput` is not `0`, `markers` **must** be a pointer to an array of `markerCapacityInput` `XrMarkerML` values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_CALL_ORDER_INVALID`

```
// Provided by XR_ML_marker_understanding
XR_DEFINE_ATOM(XrMarkerML)
```

The unique marker key used to retrieve the data about detected markers. For an `XrMarkerDetectorML` a runtime **must** use the same value of `XrMarkerML` each time a marker is detected in a snapshot, but an application **cannot** use a cached atom if it was not present in the most recent snapshot.

The `xrGetMarkerNumberML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrGetMarkerNumberML(
    XrMarkerDetectorML          markerDetector,
    XrMarkerML                  marker,
    uint64_t*                    number);
```

## Parameter Descriptions

- `markerDetector` is the detector object to retrieve marker information from.
- `marker` is the marker atom to be examined.
- `number` points to a `float` in which the numerical value associated with the marker is returned.

Get the numerical value of a marker, such as the ArUco ID. `xrGetMarkerNumberML` **must** only be called when the state of the detector is `XR_MARKER_DETECTOR_STATUS_READY_ML`. If the marker does not have an associated numerical value, the runtime **must** return `XR_ERROR_MARKER_DETECTOR_INVALID_DATA_QUERY_ML`.

If `xrGetMarkerDetectorStateML` has not been called and returned `XR_MARKER_DETECTOR_STATUS_READY_ML` since the last invocation of `xrSnapshotMarkerDetectorML`, the runtime **must** return `XR_ERROR_CALL_ORDER_INVALID`.

The runtime **must** return `XR_ERROR_MARKER_INVALID_ML` if the marker atom is invalid.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrGetMarkerNumberML`
- `markerDetector` **must** be a valid `XrMarkerDetectorML` handle
- `number` **must** be a pointer to a `uint64_t` value

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_MARKER\_INVALID\_ML
- XR\_ERROR\_MARKER\_DETECTOR\_INVALID\_DATA\_QUERY\_ML

The `xrGetMarkerStringML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrGetMarkerStringML(
    XrMarkerDetectorML    markerDetector,
    XrMarkerML            marker,
    uint32_t              bufferCapacityInput,
    uint32_t*             bufferCountOutput,
    char*                 buffer);
```



## Parameter Descriptions

- `markerDetector` is the detector object to retrieve marker information from.
- `marker` is the marker atom to be examined.
- `bufferCapacityInput` is the capacity of the buffer, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is a pointer to the count of characters written to buffer (including the terminating '\0'), or a pointer to the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an application-allocated buffer that **should** be filled with the QR code's contents. It **can** be NULL if `bufferCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

Get the string value of a marker, such as the QR encoded string. `xrCreateMarkerSpaceML` **must** only be called when the state of the detector is `XR_MARKER_DETECTOR_STATUS_READY_ML`.

If the marker does not have an associated string value, the runtime **must** return `XR_ERROR_MARKER_DETECTOR_INVALID_DATA_QUERY_ML`.

If `xrGetMarkerDetectorStateML` has not been called and returned `XR_MARKER_DETECTOR_STATUS_READY_ML` since the last invocation of `xrSnapshotMarkerDetectorML`, the runtime **must** return `XR_ERROR_CALL_ORDER_INVALID`.

This function follows the [two-call idiom](#) for filling the `buffer`.

The runtime **must** return `XR_ERROR_MARKER_INVALID_ML` if the marker atom is invalid.

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrGetMarkerStringML`
- `markerDetector` **must** be a valid `XrMarkerDetectorML` handle
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_MARKER_INVALID_ML`
- `XR_ERROR_MARKER_DETECTOR_INVALID_DATA_QUERY_ML`

The `xrGetMarkerReprojectionErrorML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrGetMarkerReprojectionErrorML(
    XrMarkerDetectorML      markerDetector,
    XrMarkerML              marker,
    float*                  reprojectionErrorMeters);
```

## Parameter Descriptions

- `markerDetector` is the detector object to retrieve marker information from.
- `marker` is the marker atom to be examined.
- `reprojectionErrorMeters` points to a `float` in which the estimated reprojection error in meters is returned.

Get the reprojection error of a marker, only available for certain types of markers. **must** only be called when the state of the detector is `XR_MARKER_DETECTOR_STATUS_READY_ML`.

If `xrGetMarkerDetectorStateML` has not been called and returned `XR_MARKER_DETECTOR_STATUS_READY_ML` since the last invocation of `xrSnapshotMarkerDetectorML`, the runtime **must** return

`XR_ERROR_CALL_ORDER_INVALID`.

A high reprojection error means that the estimated pose of the marker does not match well with the 2D detection on the processed video frame and thus the pose **may** be inaccurate. The error is given in meters, representing the displacement between real marker and its estimated pose. This means this is a normalized number, independent of marker distance or length.

The runtime **must** return `XR_ERROR_MARKER_INVALID_ML` if the marker atom is invalid.

### Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrGetMarkerReprojectionErrorML`
- `markerDetector` **must** be a valid `XrMarkerDetectorML` handle
- `reprojectionErrorMeters` **must** be a pointer to a `float` value

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_MARKER_INVALID_ML`
- `XR_ERROR_CALL_ORDER_INVALID`

The `xrGetMarkerLengthML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrGetMarkerLengthML(
    XrMarkerDetectorML          markerDetector,
    XrMarkerML                  marker,
    float*                       meters);
```

## Parameter Descriptions

- `markerDetector` is the detector object to retrieve marker information from.
- `marker` is the marker atom to be examined.
- `meters` points to a `float` in which the size per side of the queried marker is returned.

Get the size of the marker, defined as the length in meters per side. If the application created the detector while passing in a [XrMarkerDetectorSizeInfoML](#), this query **may** be redundant. [xrGetMarkerLengthML](#) is primarily intended to query for a runtime estimated size when an application did not indicate the expected size via [XrMarkerDetectorSizeInfoML](#).

[xrGetMarkerLengthML](#) **must** only be called when the state of the detector is `XR_MARKER_DETECTOR_STATUS_READY_ML`. If [xrGetMarkerDetectorStateML](#) has not been called and returned `XR_MARKER_DETECTOR_STATUS_READY_ML` since the last invocation of [xrSnapshotMarkerDetectorML](#), the runtime **must** return `XR_ERROR_CALL_ORDER_INVALID`.

The runtime **must** return `XR_ERROR_MARKER_INVALID_ML` if the marker atom is invalid.

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to calling [xrGetMarkerLengthML](#)
- `markerDetector` **must** be a valid [XrMarkerDetectorML](#) handle
- `meters` **must** be a pointer to a `float` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_MARKER_INVALID_ML`
- `XR_ERROR_CALL_ORDER_INVALID`

### 12.107.5. Getting an XrSpace from Marker Results

The `xrCreateMarkerSpaceML` function is defined as:

```
// Provided by XR_ML_marker_understanding
XrResult xrCreateMarkerSpaceML(
    XrSession session,
    const XrMarkerSpaceCreateInfoML* createInfo,
    XrSpace* space);
```

### Parameter Descriptions

- `session` is the session that will own the created space.
- `createInfo` is a pointer to the `XrMarkerSpaceCreateInfoML` used to specify the space creation parameters.
- `space` points to an `XrSpace` handle in which the resulting space is returned.

Creates an `XrSpace` from a currently snapshotted marker. The space **may** still be used even if the marker is later not in the FOV, or even if the marker detector has been destroyed. In such a scenario, the `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` and `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` **must** be

false, but `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` **may** be set as appropriate to the last known location.

Once an application has created a space, it **may** stop calling `xrSnapshotMarkerDetectorML`, and the position of the marker **must** still be updated by the runtime whenever it is aware of a more up to date location.

If a runtime is unable to spatially locate a snapshotted marker, it **may** return `XR_ERROR_MARKER_DETECTOR_LOCATE_FAILED_ML`. This is most likely to happen if significant time has passed since the snapshot of markers was acquired, and the marker in question is no longer in the user's FOV. Thus, an application **should** call `xrCreateMarkerSpaceML` immediately after examining a snapshot, but **should** also be prepared to try again if needed.

**must** only be called when the state of the detector is `XR_MARKER_DETECTOR_STATUS_READY_ML`.

If `xrGetMarkerDetectorStateML` has not been called and returned `XR_MARKER_DETECTOR_STATUS_READY_ML` since the last invocation of `xrSnapshotMarkerDetectorML`, the runtime **must** return `XR_ERROR_CALL_ORDER_INVALID`.

`session` must be the same session that created the `XrMarkerSpaceCreateInfoML::markerDetector`, else the runtime **must** return `XR_ERROR_HANDLE_INVALID`.

The runtime **must** return `XR_ERROR_MARKER_INVALID_ML` if the marker atom is invalid.

The `XrSpace` origin **must** be located at the marker's center. The X-Y plane of the `XrSpace` **must** be aligned with the plane of the marker with the positive Z axis coming out of the marker face.

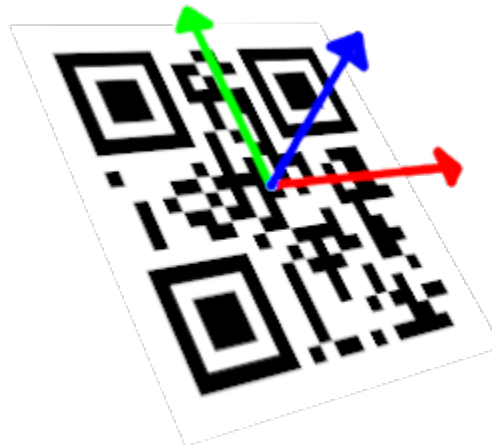


Figure 16. QR code marker with axis

## Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to calling `xrCreateMarkerSpaceML`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrMarkerSpaceCreateInfoML` structure
- `space` **must** be a pointer to an `XrSpace` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_MARKER_INVALID_ML`
- `XR_ERROR_MARKER_DETECTOR_LOCATE_FAILED_ML`
- `XR_ERROR_CALL_ORDER_INVALID`

The `XrMarkerSpaceCreateInfoML` structure is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrMarkerSpaceCreateInfoML {
    XrStructureType      type;
    const void*          next;
    XrMarkerDetectorML   markerDetector;
    XrMarkerML           marker;
    XrPosef               poseInMarkerSpace;
} XrMarkerSpaceCreateInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `markerDetector` is the detector object to retrieve marker information from.
- `marker` is the marker atom to be examined.
- `poseInMarkerSpace` is the offset from the marker's origin of the new [XrSpace](#). The origin of each marker is located at its center.

## Valid Usage (Implicit)

- The [XR\\_ML\\_marker\\_understanding](#) extension **must** be enabled prior to using [XrMarkerSpaceCreateInfoML](#)
- `type` **must** be `XR_TYPE_MARKER_SPACE_CREATE_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `markerDetector` **must** be a valid [XrMarkerDetectorML](#) handle

### 12.107.6. Example code for locating a marker

The following example code demonstrates how to detect a marker relative to a local space, and query the contents.

```
XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized
XrSpace localSpace; // previously initialized, e.g. from
                    // XR_REFERENCE_SPACE_TYPE_LOCAL
XrSpace viewSpace; // previously initialized, e.g. from
                   // XR_REFERENCE_SPACE_TYPE_VIEW
```



```

// The function pointers are previously initialized using
// xrGetInstanceProcAddr.
PFN_xrCreateMarkerDetectorML xrCreateMarkerDetectorML; // previously initialized
PFN_xrDestroyMarkerDetectorML xrDestroyMarkerDetectorML; // previously initialized
PFN_xrSnapshotMarkerDetectorML xrSnapshotMarkerDetectorML; // previously initialized
PFN_xrGetMarkerDetectorStateML xrGetMarkerDetectorStateML; // previously initialized
PFN_xrGetMarkersML xrGetMarkersML; // previously initialized
PFN_xrGetMarkerReprojectionErrorML xrGetMarkerReprojectionErrorML; // previously
initialized
PFN_xrGetMarkerLengthML xrGetMarkerLengthML; // previously initialized
PFN_xrGetMarkerNumberML xrGetMarkerNumberML; // previously initialized
PFN_xrGetMarkerStringML xrGetMarkerStringML; // previously initialized
PFN_xrCreateMarkerSpaceML xrCreateMarkerSpaceML; // previously initialized

// Initialize marker detector handle
XrMarkerDetectorML markerDetector = XR_NULL_HANDLE;

XrMarkerDetectorCreateInfoML createInfo{ XR_TYPE_MARKER_DETECTOR_CREATE_INFO_ML };
createInfo.profile = XR_MARKER_DETECTOR_PROFILE_CUSTOM_ML;
createInfo.markerType = XR_MARKER_TYPE_ARUCO_ML;

// Passing a non-custom profile allows you to leave next == nullptr
XrMarkerDetectorCustomProfileInfoML customProfile{
XR_TYPE_MARKER_DETECTOR_CUSTOM_PROFILE_INFO_ML };
customProfile.fpsHint = XR_MARKER_DETECTOR_FPS_LOW_ML;
customProfile.resolutionHint = XR_MARKER_DETECTOR_RESOLUTION_HIGH_ML;
customProfile.cameraHint = XR_MARKER_DETECTOR_CAMERA_RGB_CAMERA_ML;
customProfile.cornerRefineMethod = XR_MARKER_DETECTOR_CORNER_REFINE_METHOD_CONTOUR_ML;
customProfile.useEdgeRefinement = true;
customProfile.fullAnalysisIntervalHint =
XR_MARKER_DETECTOR_FULL_ANALYSIS_INTERVAL_SLOW_ML;

createInfo.next = &customProfile;

// Elect to use ArUco marker tracking, providing required dictionary
XrMarkerDetectorArucoInfoML arucoCreateInfo{ XR_TYPE_MARKER_DETECTOR_ARUCO_INFO_ML };
arucoCreateInfo.arucoDict = XR_MARKER_ARUCO_DICT_6X6_100_ML;

customProfile.next = &arucoCreateInfo;

// Specify the size of the marker to improve tracking quality
XrMarkerDetectorSizeInfoML sizeCreateInfo{ XR_TYPE_MARKER_DETECTOR_SIZE_INFO_ML };
sizeCreateInfo.markerLength = 0.2f;

arucoCreateInfo.next = &sizeCreateInfo;

```

```

CHK_XR(xrCreateMarkerDetectorML(session, &createInfo, &markerDetector));

bool queryRunning = false;

std::unordered_map <uint64_t, XrSpace> markerSpaceMap;

auto processMarkers = [&]() {
    // 2 call idiom to get the markers from runtime
    uint32_t markerCount;
    CHK_XR(xrGetMarkersML(markerDetector, 0, &markerCount, nullptr));
    std::vector<XrMarkerML> markers(markerCount);
    CHK_XR(xrGetMarkersML(markerDetector, markerCount, &markerCount, markers.data()));

    for(uint32_t i = 0; i < markerCount; ++i)
    {
        uint64_t number;
        CHK_XR(xrGetMarkerNumberML(markerDetector, markers[i], &number));
        // Track every marker we find.
        if(markerSpaceMap.find(number) == markerSpaceMap.end())
        {
            // New entry
            XrSpace space;
            XrMarkerSpaceCreateInfoML spaceCreateInfo{
XR_TYPE_MARKER_SPACE_CREATE_INFO_ML };
            spaceCreateInfo.markerDetector = markerDetector;
            spaceCreateInfo.marker = markers[i];
            spaceCreateInfo.poseInMarkerSpace = { {0, 0, 0, 1}, {0, 0, 0} };

            CHK_XR(xrCreateMarkerSpaceML(session, &spaceCreateInfo, &space));
            markerSpaceMap[number] = space;
        }

        // This will not work in this example with ArUco markers, but had we configured
        // a marker with string content such as QR or Code 128, this is how to use it.
        // uint32_t stringSize;
        // CHK_XR(xrGetMarkerStringML(markerDetector, markers[i], 0, &stringSize,
nullptr));
        // std::string markerString(stringSize, ' ');
        // CHK_XR(xrGetMarkerStringML(markerDetector, markers[i], stringSize,
&stringSize, markerString.data()));
    }
};

// Must be initialized to true, otherwise in the loop below, there will
// be an XR_ERROR_CALL_ORDER_INVALID due to xrSnapshotMarkerDetectorML
// not being called first
bool isReadyForSnapshot = true;

```

```

while (1) {
    // ...
    // For every frame in frame loop
    // ...

    // We have this if/else block set up so that xrSnapshotMarkerDetectorML
    // is not captured per frame since the marker detector snapshot
    // might still be in the midst of being processed by the runtime
    if (isReadyForSnapshot) {
        // Call the first snapshot
        XrMarkerDetectorSnapshotInfoML detectorInfo{
XR_TYPE_MARKER_DETECTOR_SNAPSHOT_INFO_ML };
        CHK_XR(xrSnapshotMarkerDetectorML(markerDetector, &detectorInfo));
        isReadyForSnapshot = false;
    } else {
        XrMarkerDetectorStateML state{ XR_TYPE_MARKER_DETECTOR_STATE_ML };
        CHK_XR(xrGetMarkerDetectorStateML(markerDetector, &state));
        // For simplicity, this example will assume that the marker detector will not
        // be in an erroneous state
        if (state.state == XR_MARKER_DETECTOR_STATUS_READY_ML) {
            processMarkers();
            isReadyForSnapshot = true;
        }
    }
}

// Draw the markers as needed from markerSpaceMap.
// drawMarkers(markerSpaceMap);

// ...
// ...
}
// Cleanup
CHK_XR(xrDestroyMarkerDetectorML(markerDetector));

```

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SYSTEM_MARKER_UNDERSTANDING_PROPERTIES_ML`
- `XR_TYPE_MARKER_DETECTOR_CREATE_INFO_ML`
- `XR_TYPE_MARKER_DETECTOR_ARUCO_INFO_ML`
- `XR_TYPE_MARKER_DETECTOR_APRIL_TAG_INFO_ML`
- `XR_TYPE_MARKER_DETECTOR_CUSTOM_PROFILE_INFO_ML`
- `XR_TYPE_MARKER_DETECTOR_SNAPSHOT_INFO_ML`

- `XR_TYPE_MARKER_DETECTOR_STATE_ML`
- `XR_TYPE_MARKER_SPACE_CREATE_INFO_ML`

the `XrResult` enumeration is extended with:

- `XR_ERROR_MARKER_DETECTOR_PERMISSION_DENIED_ML`
- `XR_ERROR_MARKER_DETECTOR_LOCATE_FAILED_ML`
- `XR_ERROR_MARKER_DETECTOR_INVALID_DATA_QUERY_ML`
- `XR_ERROR_MARKER_DETECTOR_INVALID_CREATE_INFO_ML`
- `XR_ERROR_MARKER_INVALID_ML`

## New Structures

The `XrSystemMarkerUnderstandingPropertiesML` structure is defined as:

```
// Provided by XR_ML_marker_understanding
typedef struct XrSystemMarkerUnderstandingPropertiesML {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsMarkerUnderstanding;
} XrSystemMarkerUnderstandingPropertiesML;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `supportsMarkerUnderstanding` indicates whether marker detection and tracking is supported by this system.

### Valid Usage (Implicit)

- The `XR_ML_marker_understanding` extension **must** be enabled prior to using `XrSystemMarkerUnderstandingPropertiesML`
- `type` **must** be `XR_TYPE_SYSTEM_MARKER_UNDERSTANDING_PROPERTIES_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## Version History

- Revision 1, 2023-05-18 (Robbie Bridgewater)
  - Initial extension skeleton

## 12.108. XR\_ML\_user\_calibration

### Name String

`XR_ML_user_calibration`

### Extension Type

Instance extension

### Registered Extension Number

473

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-08-21

### Contributors

Karthik Kadappan, Magic Leap  
Ron Bessems, Magic Leap

### 12.108.1. Overview

This extension **can** be used to determine how well the device is calibrated for the current user of the device. The extension provides two events for this purpose:

1. Headset Fit: Provides the quality of the fit of the headset on the user.
2. Eye Calibration: Provides the quality of the user's eye calibration.

### 12.108.2. Enabling user calibration events

User calibration events are requested by calling [xrEnableUserCalibrationEventsML](#). When this function is called, each of the user calibration events **must** be posted to the event queue once, regardless of whether there were any changes to the event data. This allows the application to synchronize with the current state.

The [xrEnableUserCalibrationEventsML](#) function is defined as:

```
// Provided by XR_ML_user_calibration
XrResult xrEnableUserCalibrationEventsML(
    XrInstance instance,
    const XrUserCalibrationEnableEventsInfoML* enableInfo);
```

## Parameter Descriptions

- **instance** is a handle to an [XrInstance](#) previously created with [xrCreateInstance](#).
- **enableInfo** is the [XrUserCalibrationEnableEventsInfoML](#) that enables or disables user calibration events.

## Valid Usage (Implicit)

- The [XR\\_ML\\_user\\_calibration](#) extension **must** be enabled prior to calling [xrEnableUserCalibrationEventsML](#)
- **instance** **must** be a valid [XrInstance](#) handle
- **enableInfo** **must** be a pointer to a valid [XrUserCalibrationEnableEventsInfoML](#) structure

## Return Codes

### Success

- [XR\\_SUCCESS](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)

The [XrUserCalibrationEnableEventsInfoML](#) structure is defined as:

```
// Provided by XR ML user calibration
typedef struct XrUserCalibrationEnableEventsInfoML {
    XrStructureType    type;
    const void*        next;
    XrBool32           enabled;
} XrUserCalibrationEnableEventsInfoML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `enabled` is the flag to enable/disable user calibration events.

## Valid Usage (Implicit)

- The [XR ML user calibration](#) extension **must** be enabled prior to using [XrUserCalibrationEnableEventsInfoML](#)
- `type` **must** be `XR_TYPE_USER_CALIBRATION_ENABLE_EVENTS_INFO_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.108.3. Headset Fit Events

Receiving an [XrEventDataHeadsetFitChangedML](#) event from [xrPollEvent](#) notifies the application of headset fit changes. To enable these events call [xrEnableUserCalibrationEventsML](#) and set [XrUserCalibrationEnableEventsInfoML::enabled](#) to true. Headset fit is evaluated continuously and the runtime **must** post events anytime it detects a change in the headset fit state.

The [XrEventDataHeadsetFitChangedML](#) structure is defined as:

```
// Provided by XR ML user calibration
typedef struct XrEventDataHeadsetFitChangedML {
    XrStructureType    type;
    const void*        next;
    XrHeadsetFitStatusML status;
    XrTime             time;
} XrEventDataHeadsetFitChangedML;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `status` is the [XrHeadsetFitStatusML](#) headset fit status.
- `time` is the [XrTime](#) at which the `status` was captured.

## Valid Usage (Implicit)

- The [XR\\_ML\\_user\\_calibration](#) extension **must** be enabled prior to using [XrEventDataHeadsetFitChangedML](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_HEADSET_FIT_CHANGED_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

```
// Provided by XR_ML_user_calibration
typedef enum XrHeadsetFitStatusML {
    XR_HEADSET_FIT_STATUS_UNKNOWN_ML = 0,
    XR_HEADSET_FIT_STATUS_NOT_WORN_ML = 1,
    XR_HEADSET_FIT_STATUS_GOOD_FIT_ML = 2,
    XR_HEADSET_FIT_STATUS_BAD_FIT_ML = 3,
    XR_HEADSET_FIT_STATUS_MAX_ENUM_ML = 0x7FFFFFFF
} XrHeadsetFitStatusML;
```

| Enum   | Description  |
|--|--|
| <code>XR_HEADSET_FIT_STATUS_UNKNOWN_ML</code>  | Headset fit status not available for unknown reason. |
| <code>XR_HEADSET_FIT_STATUS_NOT_WORN_ML</code> | Headset not worn.                                    |
| <code>XR_HEADSET_FIT_STATUS_GOOD_FIT_ML</code> | Good fit.  |
| <code>XR_HEADSET_FIT_STATUS_BAD_FIT_ML</code>  | Bad fit.   |

### 12.108.4. Eye Calibration Events

Receiving an [XrEventDataEyeCalibrationChangedML](#) event from [xrPollEvent](#) notifies the application of eye calibration changes. To enable these events call [xrEnableUserCalibrationEventsML](#) and set [XrUserCalibrationEnableEventsInfoML::enabled](#) to true. Runtime **must** post events anytime it detects a change in the eye calibration. The user needs to calibrate the eyes using the system app provided for



this. There is no support for in-app eye calibration in this extension.

The `XrEventDataEyeCalibrationChangedML` structure is defined as:

```
// Provided by XR_ML_user_calibration
typedef struct XrEventDataEyeCalibrationChangedML {
    XrStructureType          type;
    const void*              next;
    XrEyeCalibrationStatusML status;
} XrEventDataEyeCalibrationChangedML;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `status` is the `XrEyeCalibrationStatusML` eye calibration status.

### Valid Usage (Implicit)

- The `XR_ML_user_calibration` extension **must** be enabled prior to using `XrEventDataEyeCalibrationChangedML`
- `type` **must** be `XR_TYPE_EVENT_DATA_EYE_CALIBRATION_CHANGED_ML`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

```
// Provided by XR_ML_user_calibration
typedef enum XrEyeCalibrationStatusML {
    XR_EYE_CALIBRATION_STATUS_UNKNOWN_ML = 0,
    XR_EYE_CALIBRATION_STATUS_NONE_ML   = 1,
    XR_EYE_CALIBRATION_STATUS_COARSE_ML = 2,
    XR_EYE_CALIBRATION_STATUS_FINE_ML   = 3,
    XR_EYE_CALIBRATION_STATUS_MAX_ENUM_ML = 0x7FFFFFFF
} XrEyeCalibrationStatusML;
```

| Enum  | Description  |
|---|--|
| <code>XR_EYE_CALIBRATION_STATUS_UNKNOWN_ML</code> | Eye calibration status not available for unknown reason. |

| Enum   | Description  |
|--|--|
| <code>XR_EYE_CALIBRATION_STATUS_NONE_ML</code>   | User has not performed the eye calibration step. Use system provided app to perform eye calibration. |
| <code>XR_EYE_CALIBRATION_STATUS_COARSE_ML</code> | Eye calibration is of lower accuracy.  |
| <code>XR_EYE_CALIBRATION_STATUS_FINE_ML</code>   | Eye calibration is of higher accuracy.   |

### 12.108.5. New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_EVENT_DATA_HEADSET_FIT_CHANGED_ML`
- `XR_TYPE_EVENT_DATA_EYE_CALIBRATION_CHANGED_ML`
- `XR_TYPE_USER_CALIBRATION_ENABLE_EVENTS_INFO_ML`

#### Version History

- Revision 1, 2023-06-20 (Karthik Kadappan)
  - Initial extension description

## 12.109. XR\_MND\_headless

#### Name String

`XR_MND_headless`

#### Extension Type

Instance extension

#### Registered Extension Number

43

#### Revision

2

#### Extension and Version Dependencies

[OpenXR 1.0](#)

#### Last Modified Date

2019-10-22

#### IP Status

No known IP claims.

## Contributors

Rylie Pavlik, Collabora

## Overview

Some applications may wish to access XR interaction devices without presenting any image content on the display(s). This extension provides a mechanism for writing such an application using the OpenXR API. It modifies the specification in the following ways, without adding any new named entities.

- When this extension is enabled, an application **may** call `xrCreateSession` without an `XrGraphicsBinding*` structure in its `next` chain. In this case, the runtime **must** create a "headless" session that does not interact with the display.
- In a headless session, the session state **should** proceed to `XR_SESSION_STATE_READY` directly from `XR_SESSION_STATE_IDLE`.
- In a headless session, the `XrSessionBeginInfo::primaryViewConfigurationType` **must** be ignored and **may** be `0`.
- In a headless session, the session state proceeds to `XR_SESSION_STATE_SYNCHRONIZED`, then `XR_SESSION_STATE_VISIBLE` and `XR_SESSION_STATE_FOCUSED`, after the call to `xrBeginSession`. The application does not need to call `xrWaitFrame`, `xrBeginFrame`, or `xrEndFrame`, unlike with non-headless sessions.
- In a headless session, `xrEnumerateSwapchainFormats` **must** return `XR_SUCCESS` but enumerate `0` formats.
- `xrWaitFrame` **must** set `XrFrameState::shouldRender` to `XR_FALSE` in a headless session. The `VISIBLE` and `FOCUSED` states are only used for their input-related semantics, not their rendering-related semantics, and these functions are permitted to allow minimal change between headless and non-headless code if desired.

Because `xrWaitFrame` is not required, an application using a headless session **should** sleep periodically to avoid consuming all available system resources in a busy-wait loop.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

- Not all devices with which this would be useful fit into one of the existing `XrFormFactor` values.

## Version History

- Revision 1, 2019-07-25 (Rylie Pavlik, Collabora, Ltd.)
  - Initial version reflecting MonoD prototype.
- Revision 2, 2019-10-22 (Rylie Pavlik, Collabora, Ltd.)
  - Clarify that `xrWaitFrame` is permitted and should set `shouldRender` to false.

# 12.110. XR\_MSFT\_composition\_layer\_reprojection

## Name String

`XR_MSFT_composition_layer_reprojection`

## Extension Type

Instance extension

## Registered Extension Number

67

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2020-06-20

## IP Status

No known IP claims.

## Contributors

Zonglin Wu, Microsoft

Bryce Hutchings, Microsoft

Alex Turner, Microsoft

Yin Li, Microsoft

## Overview

This extension enables an application to provide additional reprojection information for a projection composition layer to help the runtime produce better hologram stability and visual quality.

First, the application uses [xrEnumerateReprojectionModesMSFT](#) to inspect what reprojection mode the view configuration supports.

The [xrEnumerateReprojectionModesMSFT](#) function returns the supported reprojection modes of the view configuration.

```
// Provided by XR_MSFT_composition_layer_reprojection
XrResult xrEnumerateReprojectionModesMSFT(
    XrInstance                instance,
    XrSystemId                systemId,
    XrViewConfigurationType  viewConfigurationType,
    uint32_t                  modeCapacityInput,
    uint32_t*                 modeCountOutput,
    XrReprojectionModeMSFT*  modes);
```

### Parameter Descriptions

- `instance` is the instance from which `systemId` was retrieved.
- `systemId` is the [XrSystemId](#) whose reprojection modes will be enumerated.
- `viewConfigurationType` is the [XrViewConfigurationType](#) to enumerate.
- `modeCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `modeCountOutput` is a pointer to the count of the array, or a pointer to the required capacity in the case that `modeCapacityInput` is insufficient.
- `modes` is a pointer to an application-allocated array that will be filled with the [XrReprojectionModeMSFT](#) values that are supported by the runtime. It **can** be `NULL` if `modeCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `modes` size.

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_composition\\_layer\\_reprojection](#) extension **must** be enabled prior to calling [xrEnumerateReprojectionModesMSFT](#)
- `instance` **must** be a valid [XrInstance](#) handle
- `viewConfigurationType` **must** be a valid [XrViewConfigurationType](#) value
- `modeCountOutput` **must** be a pointer to a `uint32_t` value
- If `modeCapacityInput` is not 0, `modes` **must** be a pointer to an array of `modeCapacityInput` [XrReprojectionModeMSFT](#) values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`
- `XR_ERROR_SYSTEM_INVALID`

A system **may** support different sets of reprojection modes for different view configuration types.

Then, the application **can** provide reprojection mode for the projection composition layer to inform the runtime that the XR experience **may** benefit from the provided reprojection mode.

An `XrCompositionLayerReprojectionInfoMSFT` structure **can** be added to the `next` chain of `XrCompositionLayerProjection` structure when calling `xrEndFrame`.

```
// Provided by XR_MSFT_composition_layer_reprojection
typedef struct XrCompositionLayerReprojectionInfoMSFT {
    XrStructureType      type;
    const void*          next;
    XrReprojectionModeMSFT reprojectionMode;
} XrCompositionLayerReprojectionInfoMSFT;
```

## Parameter Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `reprojectionMode` is an `XrReprojectionModeMSFT` enum providing a hint to the reprojection mode to the corresponding projection layer.

## Valid Usage (Implicit)

- The `XR_MSFT_composition_layer_reprojection` extension **must** be enabled prior to using `XrCompositionLayerReprojectionInfoMSFT`
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_REPROJECTION_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `reprojectionMode` **must** be a valid `XrReprojectionModeMSFT` value

When the application chained this structure when calling `xrEndFrame`, the `reprojectionMode` **must** be one of the supported `XrReprojectionModeMSFT` returned by `xrEnumerateReprojectionModesMSFT` function for the corresponding `XrViewConfigurationType`. Otherwise, the runtime **must** return error `XR_ERROR_REPROJECTION_MODE_UNSUPPORTED_MSFT` on the `xrEndFrame` function.

The runtime **must** only use the given information for the corresponding frame in `xrEndFrame` function, and it **must** not affect other frames.

The `XrReprojectionModeMSFT` describes the reprojection mode of a projection composition layer.

```
// Provided by XR_MSFT_composition_layer_reprojection
typedef enum XrReprojectionModeMSFT {
    XR_REPROJECTION_MODE_DEPTH_MSFT = 1,
    XR_REPROJECTION_MODE_PLANAR_FROM_DEPTH_MSFT = 2,
    XR_REPROJECTION_MODE_PLANAR_MANUAL_MSFT = 3,
    XR_REPROJECTION_MODE_ORIENTATION_ONLY_MSFT = 4,
    XR_REPROJECTION_MODE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrReprojectionModeMSFT;
```

- `XR_REPROJECTION_MODE_DEPTH_MSFT` indicates the corresponding layer **may** benefit from per-pixel depth reprojection provided by `XrCompositionLayerDepthInfoKHR` to the projection layer. This mode is typically used for world-locked content that should remain physically stationary as the user walks around.
- `XR_REPROJECTION_MODE_PLANAR_FROM_DEPTH_MSFT` indicates the corresponding layer **may** benefit from planar reprojection and the plane **can** be calculated from the corresponding depth information provided by `XrCompositionLayerDepthInfoKHR` to the projection layer. This mode works better when the application knows the content is mostly placed on a plane.
- `XR_REPROJECTION_MODE_PLANAR_MANUAL_MSFT` indicates that the corresponding layer **may** benefit from planar reprojection. The application **can** customize the plane by chaining an `XrCompositionLayerReprojectionPlaneOverrideMSFT` structure to the same layer. The app **can** also omit the plane override, indicating the runtime should use the default reprojection plane settings. This mode works better when the application knows the content is mostly placed on a plane, or when it cannot afford to submit depth information.
- `XR_REPROJECTION_MODE_ORIENTATION_ONLY_MSFT` indicates the layer should be stabilized only for changes to orientation, ignoring positional changes. This mode works better for body-locked content that should follow the user as they walk around, such as 360-degree video.

When the application passes `XR_REPROJECTION_MODE_DEPTH_MSFT` or `XR_REPROJECTION_MODE_PLANAR_FROM_DEPTH_MSFT` mode, it **should** also provide the depth buffer for the corresponding layer using `XrCompositionLayerDepthInfoKHR` in `XR_KHR_composition_layer_depth` extension. However, if the application does not submit this depth buffer, the runtime **must** apply a runtime defined fallback reprojection mode, and **must** not fail the `xrEndFrame` function because of this missing depth.

When the application passes `XR_REPROJECTION_MODE_PLANAR_MANUAL_MSFT` or `XR_REPROJECTION_MODE_ORIENTATION_ONLY_MSFT` mode, it **should** avoid providing a depth buffer for the corresponding layer using `XrCompositionLayerDepthInfoKHR` in `XR_KHR_composition_layer_depth` extension. However, if the application does submit this depth buffer, the runtime **must** not fail the `xrEndFrame` function because of this unused depth data.

When the application is confident that overriding the reprojection plane can benefit hologram stability, it **can** provide `XrCompositionLayerReprojectionPlaneOverrideMSFT` structure to further help the runtime to fine tune the reprojection details.

An application **can** add an `XrCompositionLayerReprojectionPlaneOverrideMSFT` structure to the `next` chain of `XrCompositionLayerProjection` structure.

The runtime **must** only use the given plane override for the corresponding frame in `xrEndFrame` function, and it **must** not affect other frames.



```
// Provided by XR_MSFT_composition_layer_reprojection
typedef struct XrCompositionLayerReprojectionPlaneOverrideMSFT {
    XrStructureType    type;
    const void*        next;
    XrVector3f         position;
    XrVector3f         normal;
    XrVector3f         velocity;
} XrCompositionLayerReprojectionPlaneOverrideMSFT;
```

## Parameter Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain.
- **position** describes the position of the focus plane represented in the corresponding [XrCompositionLayerProjection::space](#).
- **normal** is a unit vector describes the focus plane normal represented in the corresponding [XrCompositionLayerProjection::space](#).
- **velocity** is a velocity of the position in the corresponding [XrCompositionLayerProjection::space](#) measured in meters per second.

A runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the `normal` vector deviates by more than 1% from unit length.

Adding a reprojection plane override **may** benefit various reprojection modes including `XR_REPROJECTION_MODE_DEPTH_MSFT`, `XR_REPROJECTION_MODE_PLANAR_FROM_DEPTH_MSFT` and `XR_REPROJECTION_MODE_PLANAR_MANUAL_MSFT`.

When application choose `XR_REPROJECTION_MODE_ORIENTATION_ONLY_MSFT` mode, the reprojection plane override **may** be ignored by the runtime.

## Valid Usage (Implicit)

- The `XR_MSFT_composition_layer_reprojection` extension **must** be enabled prior to using [XrCompositionLayerReprojectionPlaneOverrideMSFT](#)
- **type** **must** be `XR_TYPE_COMPOSITION_LAYER_REPROJECTION_PLANE_OVERRIDE_MSFT`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_REPROJECTION\\_INFO\\_MSFT](#)
- [XR\\_TYPE\\_COMPOSITION\\_LAYER\\_REPROJECTION\\_PLANE\\_OVERRIDE\\_MSFT](#)

[XrResult](#) enumeration is extended with:

- [XR\\_ERROR\\_REPROJECTION\\_MODE\\_UNSUPPORTED\\_MSFT](#)

## New Enums

- [XrReprojectionModeMSFT](#)

## New Structures

- [XrCompositionLayerReprojectionInfoMSFT](#)
- [XrCompositionLayerReprojectionPlaneOverrideMSFT](#)

## New Functions

- [xrEnumerateReprojectionModesMSFT](#)

## Issues

## Version History

- Revision 1, 2020-06-20 (Yin Li)
  - Initial extension proposal

# 12.111. XR\_MSFT\_controller\_model

## Name String

[XR\\_MSFT\\_controller\\_model](#)

## Extension Type

Instance extension

## Registered Extension Number

56

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

### Contributors

Bryce Hutchings, Microsoft  
Darryl Gough, Microsoft  
Yin Li, Microsoft  
Lachlan Ford, Microsoft

### Overview

This extension provides a mechanism to load a GLTF model for controllers. An application **can** render the controller model using the real time pose input from controller's grip action pose and animate controller parts representing the user's interactions, such as pressing a button, or pulling a trigger.

This extension supports any controller interaction profile that supports *.../grip/pose*. The returned controller model represents the physical controller held in the user's hands, and it **may** be different from the current interaction profile.

### Query controller model key

[xrGetControllerModelKeyMSFT](#) retrieves the [XrControllerModelKeyMSFT](#) for a controller. This model key **may** later be used to retrieve the model data.

The [xrGetControllerModelKeyMSFT](#) function is defined as:

```
// Provided by XR_MSFT_controller_model
XrResult xrGetControllerModelKeyMSFT(
    XrSession session,
    XrPath topLevelUserPath,
    XrControllerModelKeyStateMSFT* controllerModelKeyState);
```

### Parameter Descriptions

- **session** is the specified [XrSession](#).
- **topLevelUserPath** is the top level user path corresponding to the controller render model being queried (e.g. */user/hand/left* or */user/hand/right*).
- **controllerModelKeyState** is a pointer to the [XrControllerModelKeyStateMSFT](#) to write the model key state to.

## Valid Usage (Implicit)

- The `XR_MSFT_controller_model` extension **must** be enabled prior to calling `xrGetControllerModelKeyMSFT`
- `session` **must** be a valid `XrSession` handle
- `controllerModelKeyState` **must** be a pointer to an `XrControllerModelKeyStateMSFT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_PATH_UNSUPPORTED`
- `XR_ERROR_PATH_INVALID`
- `XR_ERROR_CONTROLLER_MODEL_KEY_INVALID_MSFT`

The `XrControllerModelKeyStateMSFT` structure is defined as:

```
// Provided by XR_MSFT_controller_model
typedef struct XrControllerModelKeyStateMSFT {
    XrStructureType      type;
    void*                next;
    XrControllerModelKeyMSFT modelKey;
} XrControllerModelKeyStateMSFT;
```

## Parameter Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `modelKey` is the model key corresponding to the controller render model being queried.

The `modelKey` value for the session represents a unique controller model that can be retrieved from [xrLoadControllerModelMSFT](#) function. Therefore, the application **can** use `modelKey` to cache the returned data from [xrLoadControllerModelMSFT](#) for the session.

A `modelKey` value of [XR\\_NULL\\_CONTROLLER\\_MODEL\\_KEY\\_MSFT](#), represents an invalid model key and indicates there is no controller model yet available. The application **should** keep calling [xrGetControllerModelKeyMSFT](#) because the model **may** become available at a later point.

The returned `modelKey` value depends on an active action binding to the corresponding `.../grip/pose` of the controller. Therefore, the application **must** have provided a valid action set containing an action for `.../grip/pose`, and have successfully completed an [xrSyncActions](#) call, in order to obtain a valid `modelKey`.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_controller\\_model](#) extension **must** be enabled prior to using [XrControllerModelKeyStateMSFT](#)
- `type` **must** be [XR\\_TYPE\\_CONTROLLER\\_MODEL\\_KEY\\_STATE\\_MSFT](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

```
// Provided by XR_MSFT_controller_model
#define XR_NULL_CONTROLLER_MODEL_KEY_MSFT 0
```

[XR\\_NULL\\_CONTROLLER\\_MODEL\\_KEY\\_MSFT](#) defines an invalid model key value.

```
// Provided by XR_MSFT_controller_model
XR_DEFINE_ATOM(XrControllerModelKeyMSFT)
```

The controller model key used to retrieve the data for the renderable controller model and associated properties and state.

## Load controller model as glTF 2.0 data

Once the application obtained a valid `modelKey`, it **can** use the `xrLoadControllerModelMSFT` function to load the GLB data for the controller model.

The `xrLoadControllerModelMSFT` function loads the controller model as a byte buffer containing a binary form of glTF (a.k.a GLB file format) for the controller. The binary glTF data **must** conform to glTF 2.0 format defined at <https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html>.

```
// Provided by XR_MSFT_controller_model
XrResult xrLoadControllerModelMSFT(
    XrSession session,
    XrControllerModelKeyMSFT modelKey,
    uint32_t bufferCapacityInput,
    uint32_t* bufferCountOutput,
    uint8_t* buffer);
```

### Parameter Descriptions

- `session` is the specified `XrSession`.
- `modelKey` is the model key corresponding to the controller render model being queried.
- `bufferCapacityInput` is the capacity of the `buffer` array, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` filled in by the runtime with the count of elements in `buffer` array, or returns the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an application-allocated array of the model for the device that will be filled with the `uint8_t` values by the runtime. It **can** be `NULL` if `bufferCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

The `xrLoadControllerModelMSFT` function **may** be a slow operation and therefore **should** be invoked from a non-timing critical thread.

If the input `modelKey` is invalid, i.e. it is `XR_NULL_CONTROLLER_MODEL_KEY_MSFT` or not a key returned from `XrControllerModelKeyStateMSFT`, the runtime **must** return `XR_ERROR_CONTROLLER_MODEL_KEY_INVALID_MSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_controller_model` extension **must** be enabled prior to calling `xrLoadControllerModelMSFT`
- `session` **must** be a valid `XrSession` handle
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` `uint8_t` values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_CONTROLLER_MODEL_KEY_INVALID_MSFT`

## Animate controller parts

The application **can** animate parts of the glTF model to represent the user's interaction on the controller, such as pressing a button or pulling a trigger.

Once the application loads the glTF model of the controller, it **should** first get `XrControllerModelPropertiesMSFT` containing an array of node names in the glTF model that **can** be animated. These properties, including the order of these node names in the array, **must** be immutable for a valid `modelKey` in the session, and therefore **can** be cached. In the frame loop, the application **should** get `XrControllerModelStateMSFT` to retrieve the pose of each node representing user's interaction on the controller and apply the transform to the corresponding node in the glTF model using application's glTF renderer.

The `xrGetControllerModelPropertiesMSFT` function returns the controller model properties for a given

modelKey.

```
// Provided by XR_MSFT_controller_model
XrResult xrGetControllerModelPropertiesMSFT(
    XrSession session,
    XrControllerModelKeyMSFT modelKey,
    XrControllerModelPropertiesMSFT* properties);
```

## Parameter Descriptions

- **session** is the specified [XrSession](#).
- **modelKey** is a valid model key obtained from [XrControllerModelKeyStateMSFT](#)
- **properties** is an [XrControllerModelPropertiesMSFT](#) returning the properties of the controller model

The runtime **must** return the same data in [XrControllerModelPropertiesMSFT](#) for a valid **modelKey**. Therefore, the application **can** cache the returned [XrControllerModelPropertiesMSFT](#) using **modelKey** and reuse the data for each frame.

If the input **modelKey** is invalid, i.e. it is [XR\\_NULL\\_CONTROLLER\\_MODEL\\_KEY\\_MSFT](#) or not a key returned from [XrControllerModelKeyStateMSFT](#), the runtime **must** return [XR\\_ERROR\\_CONTROLLER\\_MODEL\\_KEY\\_INVALID\\_MSFT](#).

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_controller\\_model](#) extension **must** be enabled prior to calling [xrGetControllerModelPropertiesMSFT](#)
- **session** **must** be a valid [XrSession](#) handle
- **properties** **must** be a pointer to an [XrControllerModelPropertiesMSFT](#) structure



## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_CONTROLLER\_MODEL\_KEY\_INVALID\_MSFT

The [XrControllerModelPropertiesMSFT](#) structure describes the properties of a controller model including an array of [XrControllerModelNodePropertiesMSFT](#).

```
// Provided by XR_MSFT_controller_model
typedef struct XrControllerModelPropertiesMSFT {
    XrStructureType          type;
    void*                    next;
    uint32_t                 nodeCapacityInput;
    uint32_t                 nodeCountOutput;
    XrControllerModelNodePropertiesMSFT* nodeProperties;
} XrControllerModelPropertiesMSFT;
```

## Parameter Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `nodeCapacityInput` is the capacity of the `nodeProperties` array, or 0 to indicate a request to retrieve the required capacity.
- `nodeCountOutput` filled in by the runtime with the count of elements in `nodeProperties` array, or returns the required capacity in the case that `nodeCapacityInput` is insufficient.
- `nodeProperties` is a pointer to an application-allocated array that will be filled with the `XrControllerModelNodePropertiesMSFT` values. It **can** be `NULL` if `nodeCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `nodeProperties` size.

## Valid Usage (Implicit)

- The `XR_MSFT_controller_model` extension **must** be enabled prior to using `XrControllerModelPropertiesMSFT`
- `type` **must** be `XR_TYPE_CONTROLLER_MODEL_PROPERTIES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `nodeCapacityInput` is not 0, `nodeProperties` **must** be a pointer to an array of `nodeCapacityInput` `XrControllerModelNodePropertiesMSFT` structures

The `XrControllerModelNodePropertiesMSFT` structure describes properties of animatable nodes, including the node name and parent node name to locate a glTF node in the controller model that **can** be animated based on user's interactions on the controller.

```
// Provided by XR_MSFT_controller_model
typedef struct XrControllerModelNodePropertiesMSFT {
    XrStructureType    type;
    void*              next;
    char               parentNodeName[XR_MAX_CONTROLLER_MODEL_NODE_NAME_SIZE_MSFT];
    char               nodeName[XR_MAX_CONTROLLER_MODEL_NODE_NAME_SIZE_MSFT];
} XrControllerModelNodePropertiesMSFT;
```

## Parameter Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `parentNodeName` is the name of the parent node in the provided glTF file. The parent name **may** be empty if it should not be used to locate this node.
- `nodeName` is the name of this node in the provided glTF file.

The node can be located in the glTF node hierarchy by finding the node(s) with the matching node name and parent node name. If the `parentNodeName` is empty, the matching will be solely based on the `nodeName`.

If there are multiple nodes in the glTF file matches the condition above, the first matching node using depth-first traversal in the glTF scene **should** be animated and the rest **should** be ignored.

The runtime **must** not return any `nodeName` or `parentNodeName` that does not match any glTF nodes in the corresponding controller model.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_controller\\_model](#) extension **must** be enabled prior to using [XrControllerModelNodePropertiesMSFT](#)
- `type` **must** be `XR_TYPE_CONTROLLER_MODEL_NODE_PROPERTIES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `parentNodeName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_CONTROLLER_MODEL_NODE_NAME_SIZE_MSFT`
- `nodeName` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_CONTROLLER_MODEL_NODE_NAME_SIZE_MSFT`

The [xrGetControllerModelStateMSFT](#) function returns the current state of the controller model representing user's interaction to the controller, such as pressing a button or pulling a trigger.

```
// Provided by XR_MSFT_controller_model
XrResult xrGetControllerModelStateMSFT(
    XrSession session,
    XrControllerModelKeyMSFT modelKey,
    XrControllerModelStateMSFT* state);
```

## Parameter Descriptions

- `session` is the specified [XrSession](#).
- `modelKey` is the model key corresponding to the controller model being queried.
- `state` is a pointer to [XrControllerModelStateMSFT](#) returns the current controller model state.

The runtime **may** return different state for a model key after each call to [xrSyncActions](#), which represents the latest state of the user interactions.

If the input `modelKey` is invalid, i.e. it is [XR\\_NULL\\_CONTROLLER\\_MODEL\\_KEY\\_MSFT](#) or not a key returned from [XrControllerModelKeyStateMSFT](#), the runtime **must** return [XR\\_ERROR\\_CONTROLLER\\_MODEL\\_KEY\\_INVALID\\_MSFT](#).

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_controller\\_model](#) extension **must** be enabled prior to calling [xrGetControllerModelStateMSFT](#)
- `session` **must** be a valid [XrSession](#) handle
- `state` **must** be a pointer to an [XrControllerModelStateMSFT](#) structure

## Return Codes

### Success

- [XR\\_SUCCESS](#)
- [XR\\_SESSION\\_LOSS\\_PENDING](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_SESSION\\_LOST](#)
- [XR\\_ERROR\\_OUT\\_OF\\_MEMORY](#)
- [XR\\_ERROR\\_CONTROLLER\\_MODEL\\_KEY\\_INVALID\\_MSFT](#)

The [XrControllerModelStateMSFT](#) structure describes the state of a controller model, including an

array of [XrControllerModelNodeStateMSFT](#).

```
// Provided by XR_MSFT_controller_model
typedef struct XrControllerModelStateMSFT {
    XrStructureType          type;
    void*                    next;
    uint32_t                 nodeCapacityInput;
    uint32_t                 nodeCountOutput;
    XrControllerModelNodeStateMSFT* nodeStates;
} XrControllerModelStateMSFT;
```

### Parameter Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `nodeCapacityInput` is the capacity of the `nodeStates` array, or 0 to indicate a request to retrieve the required capacity.
- `nodeCountOutput` filled in by the runtime with the count of elements in `nodeStates` array, or returns the required capacity in the case that `nodeCapacityInput` is insufficient.
- `nodeStates` is a pointer to an application-allocated array that will be filled with the [XrControllerModelNodeStateMSFT](#) values. It **can** be `NULL` if `nodeCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `nodeStates` size.

### Valid Usage (Implicit)

- The `XR_MSFT_controller_model` extension **must** be enabled prior to using [XrControllerModelStateMSFT](#)
- `type` **must** be `XR_TYPE_CONTROLLER_MODEL_STATE_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `nodeCapacityInput` is not 0, `nodeStates` **must** be a pointer to an array of `nodeCapacityInput` [XrControllerModelNodeStateMSFT](#) structures

The [XrControllerModelNodeStateMSFT](#) structure describes the state of a node in a controller model.

```
// Provided by XR_MSFT_controller_model
typedef struct XrControllerModelNodeStateMSFT {
    XrStructureType    type;
    void*              next;
    XrPosef            nodePose;
} XrControllerModelNodeStateMSFT;
```

## Parameter Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `nodePose` is an [XrPosef](#) of the node in its parent node space.

The state is corresponding to the glTF node identified by the [XrControllerModelNodePropertiesMSFT::nodeName](#) and [XrControllerModelNodePropertiesMSFT::parentNodeName](#) of the node property at the same array index in the [XrControllerModelPropertiesMSFT::nodeProperties](#) in [XrControllerModelPropertiesMSFT](#).

The `nodePose` is based on the user's interaction on the controller at the latest [xrSyncActions](#), represented as the [XrPosef](#) of the node in its parent node space.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_controller\\_model](#) extension **must** be enabled prior to using [XrControllerModelNodeStateMSFT](#)
- `type` **must** be [XR\\_TYPE\\_CONTROLLER\\_MODEL\\_NODE\\_STATE\\_MSFT](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Object Types

## New Flag Types

## New Enum Constants

- [XR\\_MAX\\_CONTROLLER\\_MODEL\\_NODE\\_NAME\\_SIZE\\_MSFT](#)
- [XR\\_TYPE\\_CONTROLLER\\_MODEL\\_NODE\\_PROPERTIES\\_MSFT](#)
- [XR\\_TYPE\\_CONTROLLER\\_MODEL\\_PROPERTIES\\_MSFT](#)
- [XR\\_TYPE\\_CONTROLLER\\_MODEL\\_NODE\\_STATE\\_MSFT](#)
- [XR\\_TYPE\\_CONTROLLER\\_MODEL\\_STATE\\_MSFT](#)

- `XR_ERROR_CONTROLLER_MODEL_KEY_INVALID_MSFT`

## New Enums

## New Structures

- `XrControllerModelKeyStateMSFT`
- `XrControllerModelNodePropertiesMSFT`
- `XrControllerModelPropertiesMSFT`
- `XrControllerModelNodeStateMSFT`
- `XrControllerModelStateMSFT`

## New Functions

- `xrGetControllerModelKeyMSFT`
- `xrLoadControllerModelMSFT`
- `xrGetControllerModelPropertiesMSFT`
- `xrGetControllerModelStateMSFT`

## Issues

## Version History

- Revision 1, 2020-03-12 (Yin Li)
  - Initial extension description
- Revision 2, 2020-08-12 (Bryce Hutchings)
  - Remove a possible error condition

# 12.112. XR\_MSFT\_first\_person\_observer

## Name String

`XR_MSFT_first_person_observer`

## Extension Type

Instance extension

## Registered Extension Number

55

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_MSFT\\_secondary\\_view\\_configuration](#)

## Last Modified Date

2020-05-02

## IP Status

No known IP claims.

## Contributors

Yin Li, Microsoft

Zonglin Wu, Microsoft

Alex Turner, Microsoft

### 12.112.1. Overview

This first-person observer view configuration enables the runtime to request the application to render an additional first-person view of the scene to be composed onto video frames being captured from a camera attached to and moved with the primary display on the form factor, which is generally for viewing on a 2D screen by an external observer. This first-person camera will be facing forward with roughly the same perspective as the primary views, and so the application should render its view to show objects that surround the user and avoid rendering the user's body avatar. The runtime is responsible for composing the application's rendered observer view onto the camera frame based on the chosen environment blend mode for this view configuration, as this extension does not provide the associated camera frame to the application.

This extension requires the [XR\\_MSFT\\_secondary\\_view\\_configuration](#) extension to also be enabled.

`XR_VIEW_CONFIGURATION_TYPE_SECONDARY_MONO_FIRST_PERSON_OBSERVER_MSFT` requires one element in [XrViewConfigurationProperties](#) and one projection in each [XrCompositionLayerProjection](#) layer.

Runtimes **should** only make this view configuration active when the user or the application activates a runtime feature that will make use of the resulting composed camera frames, for example taking a mixed reality photo. Otherwise, the runtime **should** leave this view configuration inactive to avoid the application wasting CPU and GPU resources rendering unnecessarily for this extra view.

Because this is a first-person view of the scene, applications **can** share a common culling and instanced rendering pass with their primary view renders. However, the view state (pose and FOV) of the first-person observer view will not match the view state of any of the primary views. Applications enabling this view configuration **must** call [xrLocateViews](#) a second time each frame to explicitly query the view state for the `XR_VIEW_CONFIGURATION_TYPE_SECONDARY_MONO_FIRST_PERSON_OBSERVER_MSFT` configuration.

This secondary view configuration **may** support a different set of environment blend modes than the primary view configuration. For example, a device that only supports additive blending for its primary



display may support alpha-blending when composing the first-person observer view with camera frames. The application should render with assets and shaders that produce output acceptable to both the primary and observer view configuration's environment blend modes when sharing render passes across both view configurations.

### **New Object Types**

### **New Flag Types**

### **New Enum Constants**

[XrViewConfigurationType](#) enumeration is extended with:

- `XR_VIEW_CONFIGURATION_TYPE_SECONDARY_MONO_FIRST_PERSON_OBSERVER_MSFT`

### **New Enums**

### **New Structures**

### **New Functions**

### **Issues**

### **Version History**

- Revision 1, 2019-07-30 (Yin LI)
  - Initial extension description

## **12.113. XR\_MSFT\_hand\_interaction**

### **Name String**

`XR_MSFT_hand_interaction`

### **Extension Type**

Instance extension

### **Registered Extension Number**

51

### **Revision**

1

### **Extension and Version Dependencies**

[OpenXR 1.0](#)

## API Interactions

- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

## Contributors

Yin Li, Microsoft

Lachlan Ford, Microsoft

Alex Turner, Microsoft

## Overview

This extension defines a new interaction profile for near interactions and far interactions driven by directly-tracked hands.

## Hand interaction profile

Interaction profile path:

- */interaction\_profiles/microsoft/hand\_interaction*

### Note

The interaction profile path */interaction\_profiles/microsoft/hand\_interaction* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/microsoft/hand\_interaction\_msft*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for top level user path:

- */user/hand/left*
- */user/hand/right*

This interaction profile provides basic pose and actions for near and far interactions using hand tracking input.

Supported component paths:

- *.../input/select/value*
- *.../input/squeeze/value*
- *.../input/aim/pose*
- *.../input/grip/pose*



#### Note

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`



#### Note

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`



#### Note

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



#### Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

The application **should** use the `.../select/value` and `.../aim/pose` paths for far hand interactions, such as using a virtual laser pointer to target and click a button on the wall. Here, `.../select/value` **can** be used as either a boolean or float action type, where the value `XR_TRUE` or `1.0f` represents a closed hand shape.

The application **should** use the `.../squeeze/value` and `.../grip/pose` for near hand interactions, such as picking up a virtual object within the user's reach from a table. Here, `.../squeeze/value` **can** be used as either a boolean or float action type, where the value `XR_TRUE` or `1.0f` represents a closed hand shape.

The runtime **may** trigger both "select" and "squeeze" actions for the same hand gesture if the user's hand gesture is able to trigger both near and far interactions. The application **should** not assume they are as independent as two buttons on a controller.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

**New Structures**

**New Functions**

**Issues**

**Version History**

- Revision 1, 2019-09-16 (Yin Li)
  - Initial extension description

## 12.114. XR\_MSFT\_hand\_tracking\_mesh

**Name String**

`XR_MSFT_hand_tracking_mesh`

**Extension Type**

Instance extension

**Registered Extension Number**

53

**Revision**

4

**Extension and Version Dependencies**

[OpenXR 1.0](#)

and

[XR\\_EXT\\_hand\\_tracking](#)

**Last Modified Date**

2021-10-20

**IP Status**

No known IP claims.

**Contributors**

Yin Li, Microsoft

Lachlan Ford, Microsoft

Alex Turner, Microsoft

Bryce Hutchings, Microsoft

### 12.114.1. Overview

This extension enables hand tracking inputs represented as a dynamic hand mesh. It enables

applications to render hands in XR experiences and interact with virtual objects using hand meshes.

The application **must** also enable the [XR\\_EXT\\_hand\\_tracking](#) extension in order to use this extension.

### Inspect system capability

An application **can** inspect whether the system is capable of hand tracking meshes by chaining an [XrSystemHandTrackingMeshPropertiesMSFT](#) structure to the [XrSystemProperties](#) when calling [xrGetSystemProperties](#).

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrSystemHandTrackingMeshPropertiesMSFT {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsHandTrackingMesh;
    uint32_t           maxHandMeshIndexCount;
    uint32_t           maxHandMeshVertexCount;
} XrSystemHandTrackingMeshPropertiesMSFT;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsHandTrackingMesh` is an [XrBool32](#), indicating if current system is capable of hand tracking mesh input.
- `maxHandMeshIndexCount` is a [uint32\\_t](#) returns the maximum count of indices that will be returned from the hand tracker.
- `maxHandMeshVertexCount` is a [uint32\\_t](#) returns the maximum count of vertices that will be returned from the hand tracker.

If a runtime returns `XR_FALSE` for `supportsHandTrackingMesh`, the system does not support hand tracking mesh input, and therefore **must** return `XR_ERROR_FEATURE_UNSUPPORTED` from [xrCreateHandMeshSpaceMSFT](#) and [xrUpdateHandMeshMSFT](#). The application **should** avoid using hand mesh functionality when `supportsHandTrackingMesh` is `XR_FALSE`.

If a runtime returns `XR_TRUE` for `supportsHandTrackingMesh`, the system supports hand tracking mesh input. In this case, the runtime **must** return a positive number for `maxHandMeshIndexCount` and `maxHandMeshVertexCount`. An application **should** use `maxHandMeshIndexCount` and `maxHandMeshVertexCount` to preallocate hand mesh buffers and reuse them in their render loop when calling [xrUpdateHandMeshMSFT](#) every frame.

## Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to using `XrSystemHandTrackingMeshPropertiesMSFT`
- `type` **must** be `XR_TYPE_SYSTEM_HAND_TRACKING_MESH_PROPERTIES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

### 12.114.2. Obtain a hand tracker handle

An application first creates an `XrHandTrackerEXT` handle using the `xrCreateHandTrackerEXT` function for each hand. The application can also reuse the same `XrHandTrackerEXT` handle previously created for the hand joint tracking. When doing so, the hand mesh input is always in sync with hand joints input with the same `XrHandTrackerEXT` handle.

### 12.114.3. Create a hand mesh space

The application creates a hand mesh space using function `xrCreateHandMeshSpaceMSFT`. The position and normal of hand mesh vertices will be represented in this space.

```
// Provided by XR_MSFT_hand_tracking_mesh
XrResult xrCreateHandMeshSpaceMSFT(
    XrHandTrackerEXT          handTracker,
    const XrHandMeshSpaceCreateInfoMSFT* createInfo,
    XrSpace*                  space);
```

## Parameter Descriptions

- `handTracker` is an `XrHandTrackerEXT` handle previously created with the `xrCreateHandTrackerEXT` function.
- `createInfo` is the `XrHandMeshSpaceCreateInfoMSFT` used to specify the hand mesh space.
- `space` is the returned `XrSpace` handle of the new hand mesh space.

A hand mesh space location is specified by runtime preference to effectively represent hand mesh vertices without unnecessary transformations. For example, an optical hand tracking system **can** define the hand mesh space origin at the depth camera's optical center.

An application should create separate hand mesh space handles for each hand to retrieve the corresponding hand mesh data. The runtime **may** use the lifetime of this hand mesh space handle to manage the underlying device resources. Therefore, the application **should** destroy the hand mesh

handle after it is finished using the hand mesh.

The hand mesh space can be related to other spaces in the session, such as view reference space, or grip action space from the `/interaction_profiles/khr/simple_controller` interaction profile. The hand mesh space may be not locatable when the hand is outside of the tracking range, or if focus is removed from the application. In these cases, the runtime **must** not set the `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` bits on calls to `xrLocateSpace` with the hand mesh space, and the application **should** avoid using the returned poses or query for hand mesh data.

If the underlying `XrHandTrackerEXT` is destroyed, the runtime **must** continue to support `xrLocateSpace` using the hand mesh space, and it **must** return space location with `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_SPACE_LOCATION_ORIENTATION_VALID_BIT` unset.

The application **may** create a mesh space for the reference hand by setting `XrHandPoseTypeInfoMSFT::handPoseType` to `XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT`. Hand mesh spaces for the reference hand **must** only be locatable in reference to mesh spaces or joint spaces of the reference hand.

### Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to calling `xrCreateHandMeshSpaceMSFT`
- `handTracker` **must** be a valid `XrHandTrackerEXT` handle
- `createInfo` **must** be a pointer to a valid `XrHandMeshSpaceCreateInfoMSFT` structure
- `space` **must** be a pointer to an `XrSpace` handle

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_LIMIT\_REACHED
- XR\_ERROR\_POSE\_INVALID
- XR\_ERROR\_FEATURE\_UNSUPPORTED

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandMeshSpaceCreateInfoMSFT {
    XrStructureType      type;
    const void*          next;
    XrHandPoseTypeMSFT  handPoseType;
    XrPosef              poseInHandMeshSpace;
} XrHandMeshSpaceCreateInfoMSFT;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `handPoseType` is an [XrHandPoseTypeMSFT](#) used to specify the type of hand this mesh is tracking. Indices and vertices returned from [xrUpdateHandMeshMSFT](#) for a hand type will be relative to the corresponding space create with the same hand type.
- `poseInHandMeshSpace` is an [XrPosef](#) defining the position and orientation of the new space's origin within the natural reference frame of the hand mesh space.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_hand\\_tracking\\_mesh](#) extension **must** be enabled prior to using [XrHandMeshSpaceCreateInfoMSFT](#)
- `type` **must** be `XR_TYPE_HAND_MESH_SPACE_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `handPoseType` **must** be a valid [XrHandPoseTypeMSFT](#) value

### 12.114.4. Locate the hand mesh

The application **can** use the [xrUpdateHandMeshMSFT](#) function to retrieve the hand mesh at a given timestamp. The hand mesh's vertices position and normal are represented in the hand mesh space created by [xrCreateHandMeshSpaceMSFT](#) with a same [XrHandTrackerEXT](#).

```
// Provided by XR_MSFT_hand_tracking_mesh
XrResult xrUpdateHandMeshMSFT(
    XrHandTrackerEXT                handTracker,
    const XrHandMeshUpdateInfoMSFT* updateInfo,
    XrHandMeshMSFT*                 handMesh);
```

## Parameter Descriptions

- `handTracker` is an `XrHandTrackerEXT` handle previously created with `xrCreateHandTrackerEXT`.
- `updateInfo` is an `XrHandMeshUpdateInfoMSFT` which contains information to query the hand mesh.
- `handMesh` is an `XrHandMeshMSFT` structure to receive the updates of hand mesh data.

The application **should** preallocate the index buffer and vertex buffer in `XrHandMeshMSFT` using the `XrSystemHandTrackingMeshPropertiesMSFT::maxHandMeshIndexCount` and `XrSystemHandTrackingMeshPropertiesMSFT::maxHandMeshVertexCount` from the `XrSystemHandTrackingMeshPropertiesMSFT` returned from the `xrGetSystemProperties` function.

The application **should** preallocate the `XrHandMeshMSFT` structure and reuse it for each frame so as to reduce the copies of data when underlying tracking data is not changed. The application should use `XrHandMeshMSFT::indexBufferChanged` and `XrHandMeshMSFT::vertexBufferChanged` in `XrHandMeshMSFT` to detect changes and avoid unnecessary data processing when there is no changes.

## Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to calling `xrUpdateHandMeshMSFT`
- `handTracker` **must** be a valid `XrHandTrackerEXT` handle
- `updateInfo` **must** be a pointer to a valid `XrHandMeshUpdateInfoMSFT` structure
- `handMesh` **must** be a pointer to an `XrHandMeshMSFT` structure

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_TIME\_INVALID
- XR\_ERROR\_FEATURE\_UNSUPPORTED

A [XrHandMeshUpdateInfoMSFT](#) describes the information to update a hand mesh.

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandMeshUpdateInfoMSFT {
    XrStructureType      type;
    const void*          next;
    XrTime               time;
    XrHandPoseTypeMSFT  handPoseType;
} XrHandMeshUpdateInfoMSFT;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **time** is the [XrTime](#) that describes the time for which the application wishes to query the hand mesh state.
- **handPoseType** is an [XrHandPoseTypeMSFT](#) which describes the type of hand pose of the hand mesh to update.

A runtime **may** not maintain a full history of hand mesh data, therefore the returned [XrHandMeshMSFT](#) might return data that's not exactly corresponding to the **time** input. If the runtime cannot return any tracking data for the given **time** at all, it **must** set [XrHandMeshMSFT::isActive](#) to `XR_FALSE` for the call to [xrUpdateHandMeshMSFT](#). Otherwise, if the runtime returns [XrHandMeshMSFT::isActive](#) as `XR_TRUE`, the data in [XrHandMeshMSFT](#) must be valid to use.

An application can choose different **handPoseType** values to query the hand mesh data. The returned hand mesh **must** be consistent to the hand joint space location on the same [XrHandTrackerEXT](#) when using the same [XrHandPoseTypeMSFT](#).

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_hand\\_tracking\\_mesh](#) extension **must** be enabled prior to using [XrHandMeshUpdateInfoMSFT](#)
- **type** **must** be `XR_TYPE_HAND_MESH_UPDATE_INFO_MSFT`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **handPoseType** **must** be a valid [XrHandPoseTypeMSFT](#) value

A [XrHandMeshMSFT](#) structure contains data and buffers to receive updates of hand mesh tracking data from [xrUpdateHandMeshMSFT](#) function.

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandMeshMSFT {
    XrStructureType          type;
    void*                    next;
    XrBool32                 isActive;
    XrBool32                 indexBufferChanged;
    XrBool32                 vertexBufferChanged;
    XrHandMeshIndexBufferMSFT indexBuffer;
    XrHandMeshVertexBufferMSFT vertexBuffer;
} XrHandMeshMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `isActive` is an `XrBool32` indicating if the current hand tracker is active.
- `indexBufferChanged` is an `XrBool32` indicating if the `indexBuffer` content was changed during the update.
- `vertexBufferChanged` is an `XrBool32` indicating if the `vertexBuffer` content was changed during the update.
- `indexBuffer` is an `XrHandMeshIndexBufferMSFT` returns the index buffer of the tracked hand mesh.
- `vertexBuffer` is an `XrHandMeshVertexBufferMSFT` returns the vertex buffer of the tracked hand mesh.

When the returned `isActive` value is `XR_FALSE`, the runtime indicates the hand is not actively tracked, for example, the hand is outside of sensor’s range, or the input focus is taken away from the application. When the runtime returns `XR_FALSE` to `isActive`, it **must** set `indexBufferChanged` and `vertexBufferChanged` to `XR_FALSE`, and **must** not change the content in `indexBuffer` or `vertexBuffer`,

When the returned `isActive` value is `XR_TRUE`, the hand tracking mesh represented in `indexBuffer` and `vertexBuffer` are updated to the latest data of the `XrHandMeshUpdateInfoMSFT::time` given to the `xrUpdateHandMeshMSFT` function. The runtime **must** set `indexBufferChanged` and `vertexBufferChanged` to reflect whether the index or vertex buffer’s content are changed during the update. In this way, the application can easily avoid unnecessary processing of buffers when there’s no new data.

The hand mesh is represented in triangle lists and each triangle’s vertices are in clockwise order when looking from outside of the hand. When hand tracking is active, i.e. when `isActive` is returned as `XR_TRUE`, the returned `indexBuffer.indexCountOutput` value **must** be positive and multiple of 3, and `vertexBuffer.vertexCountOutput` value **must** be equal to or larger than 3.

## Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to using `XrHandMeshMSFT`
- `type` **must** be `XR_TYPE_HAND_MESH_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `indexBuffer` **must** be a valid `XrHandMeshIndexBufferMSFT` structure
- `vertexBuffer` **must** be a valid `XrHandMeshVertexBufferMSFT` structure

A `XrHandMeshIndexBufferMSFT` structure includes an array of indices describing the triangle list of a hand mesh.

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandMeshIndexBufferMSFT {
    uint32_t    indexBufferKey;
    uint32_t    indexCapacityInput;
    uint32_t    indexCountOutput;
    uint32_t*   indices;
} XrHandMeshIndexBufferMSFT;
```

## Member Descriptions

- `indexBufferKey` is a `uint32_t` serving as the key of the returned index buffer content or 0 to indicate a request to retrieve the latest indices regardless of existing content in `indices`.
- `indexCapacityInput` is a positive `uint32_t` describes the capacity of the `indices` array.
- `indexCountOutput` is a `uint32_t` returned by the runtime with the count of indices written in `indices`.
- `indices` is an array of indices filled in by the runtime, specifying the indices of the triangles list in the vertex buffer.

An application **should** preallocate the `indices` array using the `XrSystemHandTrackingMeshPropertiesMSFT::maxHandMeshIndexCount` returned from `xrGetSystemProperties`. In this way, the application can avoid possible insufficient buffer sizes for each query, and therefore avoid reallocating memory each frame.

The input `indexCapacityInput` **must** not be 0, and `indices` **must** not be `NULL`, or else the runtime **must** return `XR_ERROR_VALIDATION_FAILURE` on calls to the `xrUpdateHandMeshMSFT` function.

If the input `indexCapacityInput` is not sufficient to contain all output indices, the runtime **must** return

`XR_ERROR_SIZE_INSUFFICIENT` on calls to `xrUpdateHandMeshMSFT`, not change the content in `indexBufferKey` and `indices`, and return 0 for `indexCountOutput`.

If the input `indexCapacityInput` is equal to or larger than the `XrSystemHandTrackingMeshPropertiesMSFT::maxHandMeshIndexCount` returned from `xrGetSystemProperties`, the runtime **must** not return `XR_ERROR_SIZE_INSUFFICIENT` error on `xrUpdateHandMeshMSFT` because of insufficient index buffer size.

If the input `indexBufferKey` is 0, the capacity of indices array is sufficient, and hand mesh tracking is active, the runtime **must** return the latest non-zero `indexBufferKey`, and fill in `indexCountOutput` and `indices`.

If the input `indexBufferKey` is not 0, the runtime **can** either return without changing `indexCountOutput` or content in `indices`, and return `XR_FALSE` for `XrHandMeshMSFT::indexBufferChanged` indicating the indices are not changed; or return a new non-zero `indexBufferKey` and fill in latest data in `indexCountOutput` and `indices`, and return `XR_TRUE` for `XrHandMeshMSFT::indexBufferChanged` indicating the indices are updated to a newer version.

An application **can** keep the `XrHandMeshIndexBufferMSFT` structure for each frame in a frame loop and use the returned `indexBufferKey` to identify different triangle list topology described in `indices`. The application can therefore avoid unnecessary processing of indices, such as coping them to GPU memory.

The runtime **must** return the same `indexBufferKey` for the same `XrHandTrackerEXT` at a given time, regardless of the input `XrHandPoseTypeMSFT` in `XrHandMeshUpdateInfoMSFT`. This ensures the index buffer has the same mesh topology and allows the application to reason about vertices across different hand pose types. For example, the application **can** build a procedure to perform UV mapping on vertices of a hand mesh using `XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT`, and apply the resultant UV data on vertices to the mesh returned from the same hand tracker using `XR_HAND_POSE_TYPE_TRACKED_MSFT`.

### Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to using `XrHandMeshIndexBufferMSFT`
- If `indexCapacityInput` is not 0, `indices` **must** be a pointer to an array of `indexCapacityInput` `uint32_t` values

A `XrHandMeshVertexBufferMSFT` structure includes an array of vertices of the hand mesh represented in the hand mesh space.

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandMeshVertexBufferMSFT {
    XrTime                vertexUpdateTime;
    uint32_t              vertexCapacityInput;
    uint32_t              vertexCountOutput;
    XrHandMeshVertexMSFT* vertices;
} XrHandMeshVertexBufferMSFT;
```

## Member Descriptions

- `vertexUpdateTime` is an `XrTime` representing the time when the runtime receives the vertex buffer content or 0 to indicate a request to retrieve latest vertices regardless of existing content in `vertices`.
- `vertexCapacityInput` is a positive `uint32_t` describes the capacity of the `vertices` array.
- `vertexCountOutput` is a `uint32_t` filled in by the runtime with the count of vertices written in `vertices`.
- `vertices` is an array of `XrHandMeshVertexMSFT` filled in by the runtime, specifying the vertices of the hand mesh including the position and normal vector in the hand mesh space.

An application **should** preallocate the `vertices` array using the `XrSystemHandTrackingMeshPropertiesMSFT::maxHandMeshVertexCount` returned from `xrGetSystemProperties`. In this way, the application can avoid possible insufficient buffer sizes for each query, and therefore avoid reallocating memory each frame.

The input `vertexCapacityInput` **must** not be 0, and `vertices` **must** not be `NULL`, or else the runtime **must** return `XR_ERROR_VALIDATION_FAILURE` on calls to the `xrUpdateHandMeshMSFT` function.

If the input `vertexCapacityInput` is not sufficient to contain all output vertices, the runtime **must** return `XR_ERROR_SIZE_INSUFFICIENT` on calls to the `xrUpdateHandMeshMSFT`, do not change content in `vertexUpdateTime` and `vertices`, and return 0 for `vertexCountOutput`.

If the input `vertexCapacityInput` is equal to or larger than the `XrSystemHandTrackingMeshPropertiesMSFT::maxHandMeshVertexCount` returned from `xrGetSystemProperties`, the runtime **must** not return `XR_ERROR_SIZE_INSUFFICIENT` on calls to the `xrUpdateHandMeshMSFT` because of insufficient vertex buffer size.

If the input `vertexUpdateTime` is 0, and the capacity of the `vertices` array is sufficient, and hand mesh tracking is active, the runtime **must** return the latest non-zero `vertexUpdateTime`, and fill in the `vertexCountOutput` and `vertices` fields.

If the input `vertexUpdateTime` is not 0, the runtime **can** either return without changing `vertexCountOutput` or the content in `vertices`, and return `XR_FALSE` for `XrHandMeshMSFT::vertexBufferChanged` indicating the vertices are not changed; or return a new non-zero



`vertexUpdateTime` and fill in latest data in `vertexCountOutput` and `vertices` and return `XR_TRUE` for `XrHandMeshMSFT::vertexBufferChanged` indicating the vertices are updated to a newer version.

An application **can** keep the `XrHandMeshVertexBufferMSFT` structure for each frame in frame loop and use the returned `vertexUpdateTime` to detect the changes of the content in `vertices`. The application can therefore avoid unnecessary processing of vertices, such as coping them to GPU memory.

### Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to using `XrHandMeshVertexBufferMSFT`
- If `vertexCapacityInput` is not 0, `vertices` **must** be a pointer to an array of `vertexCapacityInput` `XrHandMeshVertexMSFT` structures

Each `XrHandMeshVertexMSFT` includes the position and normal of a vertex of a hand mesh.

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandMeshVertexMSFT {
    XrVector3f    position;
    XrVector3f    normal;
} XrHandMeshVertexMSFT;
```

### Member Descriptions

- `position` is an `XrVector3f` structure representing the position of the vertex in the hand mesh space, measured in meters.
- `normal` is an `XrVector3f` structure representing the unweighted normal of the triangle surface at the vertex as a unit vector in hand mesh space.

### Valid Usage (Implicit)

- The `XR_MSFT_hand_tracking_mesh` extension **must** be enabled prior to using `XrHandMeshVertexMSFT`

## 12.114.5. Example code for hand mesh tracking

Following example code demos preallocating hand mesh buffers and updating the hand mesh in rendering loop

```

XrInstance instance; // previously initialized
XrSystemId systemId; // previously initialized
XrSession session; // previously initialized

// Inspect hand tracking mesh system properties
XrSystemHandTrackingMeshPropertiesMSFT
handMeshSystemProperties{XR_TYPE_SYSTEM_HAND_TRACKING_MESH_PROPERTIES_MSFT};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES,
&handMeshSystemProperties};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));
if (!handMeshSystemProperties.supportsHandTrackingMesh) {
    // the system does not support hand mesh tracking
    return;
}

// Get function pointer for xrCreateHandTrackerEXT
PFN_xrCreateHandTrackerEXT pfnCreateHandTrackerEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateHandTrackerEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
&pfnCreateHandTrackerEXT)));

// Create a tracker for left hand.
XrHandTrackerEXT leftHandTracker{};
{
    XrHandTrackerCreateInfoEXT createInfo{XR_TYPE_HAND_TRACKER_CREATE_INFO_EXT};
    createInfo.hand = XR_HAND_LEFT_EXT;
    createInfo.handJointSet = XR_HAND_JOINT_SET_DEFAULT_EXT;
    CHK_XR(pfnCreateHandTrackerEXT(session, &createInfo, &leftHandTracker));
}

// Get function pointer for xrCreateHandMeshSpaceMSFT
PFN_xrCreateHandMeshSpaceMSFT pfnCreateHandMeshSpaceMSFT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateHandMeshSpaceMSFT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
&pfnCreateHandMeshSpaceMSFT)));

// Create the hand mesh spaces
XrSpace leftHandMeshSpace{};
{
    XrHandMeshSpaceCreateInfoMSFT createInfo{XR_TYPE_HAND_MESH_SPACE_CREATE_INFO_MSFT};
    createInfo.poseInHandMeshSpace = {{0, 0, 0, 1}, {0, 0, 0}};
    CHK_XR(pfnCreateHandMeshSpaceMSFT(leftHandTracker, &createInfo, &leftHandMeshSpace));
}

// Preallocate buffers for hand mesh indices and vertices
std::vector<uint32_t> handMeshIndices(handMeshSystemProperties.maxHandMeshIndexCount);
std::vector<XrHandMeshVertexMSFT>
handMeshVertices(handMeshSystemProperties.maxHandMeshVertexCount);

```

```

XrHandMeshMSFT leftHandMesh{XR_TYPE_HAND_MESH_MSFT};
leftHandMesh.indexBuffer.indexCapacityInput = (uint32_t)handMeshIndices.size();
leftHandMesh.indexBuffer.indices = handMeshIndices.data();
leftHandMesh.vertexBuffer.vertexCapacityInput = (uint32_t)handMeshVertices.size();
leftHandMesh.vertexBuffer.vertices = handMeshVertices.data();

// Get function pointer for xrUpdateHandMeshMSFT
PFN_xrUpdateHandMeshMSFT pfnUpdateHandMeshMSFT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrUpdateHandMeshMSFT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnUpdateHandMeshMSFT)));

while(1){
    // ...
    // For every frame in frame loop
    // ...
    XrFrameState frameState;    // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrHandMeshUpdateInfoMSFT updateInfo{XR_TYPE_HAND_MESH_UPDATE_INFO_MSFT};
    updateInfo.time = time;
    CHK_XR(pfnUpdateHandMeshMSFT(leftHandTracker, &updateInfo, &leftHandMesh));
    if (!leftHandMesh.isActive) {
        // Hand input is not focused or user's hand is out of tracking range.
        // Do not process or render hand mesh.
    } else {
        if (leftHandMesh.indexBufferChanged) {
            // Process indices in indexBuffer.indices
        }

        if (leftHandMesh.vertexBufferChanged) {
            // Process vertices in vertexBuffer.vertices and leftHandMeshSpace
        }
    }
}
}

```

### 12.114.6. Get hand reference poses

By default, an [XrHandTrackerEXT](#) tracks a default hand pose type, that is to provide best fidelity to the user's actual hand motion. This is the same with [XR\\_HAND\\_POSE\\_TYPE\\_TRACKED\\_MSFT](#) (i.e. value 0) in a chained [XrHandPoseTypeInfoMSFT](#) structure to the `next` pointer of [XrHandTrackerCreateInfoEXT](#) when calling [xrCreateHandTrackerEXT](#).

Some hand mesh visualizations may require an initial analysis or processing of the hand mesh relative to the joints of the hand. For example, a hand visualization may generate a UV mapping for the hand mesh vertices by raycasting outward from key joints against the mesh to find key vertices.

To avoid biasing such static analysis with the arbitrary tracked hand pose, an application **can** instead create a different [XrHandTrackerEXT](#) handle with a reference hand pose type when calling [xrCreateHandTrackerEXT](#). This will instruct the runtime to provide a reference hand pose that is better suited for such static analysis.

An application can chain an [XrHandPoseTypeInfoMSFT](#) structure to the [XrHandTrackerCreateInfoEXT::next](#) pointer when calling [xrCreateHandTrackerEXT](#) to indicate the hand tracker to return the hand pose of specific [XrHandPoseTypeMSFT](#).

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef struct XrHandPoseTypeInfoMSFT {
    XrStructureType    type;
    const void*        next;
    XrHandPoseTypeMSFT handPoseType;
} XrHandPoseTypeInfoMSFT;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- **handPoseType** is an [XrHandPoseTypeMSFT](#) that describes the type of hand pose of the hand tracking.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_hand\\_tracking\\_mesh](#) extension **must** be enabled prior to using [XrHandPoseTypeInfoMSFT](#)
- **type** **must** be `XR_TYPE_HAND_POSE_TYPE_INFO_MSFT`
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **handPoseType** **must** be a valid [XrHandPoseTypeMSFT](#) value

The [XrHandPoseTypeMSFT](#) describes the type of input hand pose from [XrHandTrackerEXT](#).

```
// Provided by XR_MSFT_hand_tracking_mesh
typedef enum XrHandPoseTypeMSFT {
    XR_HAND_POSE_TYPE_TRACKED_MSFT = 0,
    XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT = 1,
    XR_HAND_POSE_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrHandPoseTypeMSFT;
```

## Enumerant Descriptions

- `XR_HAND_POSE_TYPE_TRACKED_MSFT` represents a hand pose provided by actual tracking of the user's hand.
- `XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT` represents a stable reference hand pose in a relaxed open hand shape.

The `XR_HAND_POSE_TYPE_TRACKED_MSFT` input provides best fidelity to the user's actual hand motion. When the hand tracking input requires the user to be holding a controller in their hand, the hand tracking input will appear as the user virtually holding the controller. This input can be used to render the hand shape together with the controller in hand.

The `XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT` input does not move with the user's actual hand. Through this reference hand pose, an application **can** get a stable hand joint and mesh that has the same mesh topology as the tracked hand mesh using the same `XrHandTrackerEXT`, so that the application can apply the data computed from a reference hand pose to the corresponding tracked hand.

Although a reference hand pose does not move with user's hand motion, the bone length and hand thickness **may** be updated, for example when tracking result refines, or a different user's hand is detected. The application **should** update reference hand joints and meshes when the tracked mesh's `indexBufferKey` is changed or when the `isActive` value returned from `xrUpdateHandMeshMSFT` changes from `XR_FALSE` to `XR_TRUE`. It can use the returned `indexBufferKey` and `vertexUpdateTime` from `xrUpdateHandMeshMSFT` to avoid unnecessary CPU or GPU work to process the neutral hand inputs.

### 12.114.7. Example code for reference hand mesh update

The following example code demonstrates detecting reference hand mesh changes and retrieving data for processing.

```
XrInstance instance;           // previously initialized
XrSession session;           // previously initialized
XrHandTrackerEXT handTracker; // previously initialized with handJointSet set to
XR_HAND_JOINT_SET_DEFAULT_MSFT
XrSpace handMeshReferenceSpace; // previously initialized with handPoseType set to
XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT
```

```

XrHandMeshMSFT referenceHandMesh; // previously initialized with preallocated buffers

// Get function pointer for xrUpdateHandMeshMSFT
PFN_xrUpdateHandMeshMSFT pfnUpdateHandMeshMSFT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrUpdateHandMeshMSFT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnUpdateHandMeshMSFT)));

// Get function pointer for xrCreateHandTrackerEXT
PFN_xrCreateHandTrackerEXT pfnCreateHandTrackerEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrCreateHandTrackerEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnCreateHandTrackerEXT)));

// Get function pointer for xrLocateHandJointsEXT
PFN_xrLocateHandJointsEXT pfnLocateHandJointsEXT;
CHK_XR(xrGetInstanceProcAddr(instance, "xrLocateHandJointsEXT",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &pfnLocateHandJointsEXT)));

while(1){
    // ...
    // For every frame in frame loop
    // ...
    XrFrameState frameState; // previously returned from xrWaitFrame
    const XrTime time = frameState.predictedDisplayTime;

    XrHandMeshUpdateInfoMSFT updateInfo{XR_TYPE_HAND_MESH_UPDATE_INFO_MSFT};
    updateInfo.time = time;
    updateInfo.handPoseType = XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT;
    CHK_XR(pfnUpdateHandMeshMSFT(handTracker, &updateInfo, &referenceHandMesh));

    // Detect if reference hand mesh is changed.
    if (referenceHandMesh.indexBufferChanged || referenceHandMesh.vertexBufferChanged) {

        // Query the joint location using "open palm" reference hand pose.
        XrHandPoseTypeInfoMSFT handPoseTypeInfo{XR_TYPE_HAND_POSE_TYPE_INFO_MSFT};
        handPoseTypeInfo.handPoseType = XR_HAND_POSE_TYPE_REFERENCE_OPEN_PALM_MSFT;

        XrHandTrackerCreateInfoEXT createInfo{XR_TYPE_HAND_TRACKER_CREATE_INFO_EXT};
        createInfo.hand = XR_HAND_LEFT_EXT;
        createInfo.handJointSet = XR_HAND_JOINT_SET_DEFAULT_EXT;
        createInfo.next = &handPoseTypeInfo;

        XrHandTrackerEXT referenceHandTracker;
        CHK_XR(pfnCreateHandTrackerEXT(session, &createInfo, &referenceHandTracker));

        XrHandJointsLocateInfoEXT locateInfo{XR_TYPE_HAND_JOINTS_LOCATE_INFO_EXT};
        locateInfo.next = &handPoseTypeInfo;
    }
}

```

```

    locateInfo.baseSpace = handMeshReferenceSpace; // Query joint location relative
to hand mesh reference space
    locateInfo.time = time;

    std::array<XrHandJointLocationEXT, XR_HAND_JOINT_COUNT_EXT> jointLocations;
    XrHandJointLocationsEXT locations{XR_TYPE_HAND_JOINT_LOCATIONS_EXT};
    locations.jointCount = jointLocations.size();
    locations.jointLocations = jointLocations.data();

    CHK_XR(pfnLocateHandJointsEXT(referenceHandTracker, &locateInfo, &locations));

    // Generate UV map using tip/wrist location and referenceHandMesh.vertexBuffer
    // For example, gradually changes color from the tip of the hand to wrist.
}
}

```

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_HAND\\_MESH\\_SPACE\\_CREATE\\_INFO\\_MSFT](#)
- [XR\\_TYPE\\_HAND\\_MESH\\_UPDATE\\_INFO\\_MSFT](#)
- [XR\\_TYPE\\_HAND\\_MESH\\_MSFT](#)
- [XR\\_TYPE\\_SYSTEM\\_HAND\\_TRACKING\\_MESH\\_PROPERTIES\\_MSFT](#)
- [XR\\_TYPE\\_HAND\\_POSE\\_TYPE\\_INFO\\_MSFT](#)

## New Enums

- [XrHandPoseTypeMSFT](#)

## New Structures

- [XrHandMeshSpaceCreateInfoMSFT](#)
- [XrHandMeshUpdateInfoMSFT](#)
- [XrHandMeshMSFT](#)
- [XrHandMeshIndexBufferMSFT](#)
- [XrHandMeshVertexBufferMSFT](#)
- [XrHandMeshVertexMSFT](#)
- [XrSystemHandTrackingMeshPropertiesMSFT](#)

- [XrHandPoseTypeInfoMSFT](#)

## New Functions

- [xrCreateHandMeshSpaceMSFT](#)
- [xrUpdateHandMeshMSFT](#)

## Issues

## Version History

- Revision 1, 2019-09-20 (Yin LI)
  - Initial extension description
- Revision 2, 2020-04-20 (Yin LI)
  - Change joint spaces to locate joints function.
- Revision 3, 2021-04-13 (Rylie Pavlik, Collabora, Ltd.)
  - Correctly show function pointer retrieval in sample code
- Revision 4, 2021-10-20 (Darryl Gough)
  - Winding order for hand mesh is corrected to clockwise to match runtime behavior.

# 12.115. XR\_MSFT\_holographic\_window\_attachment

## Name String

`XR_MSFT_holographic_window_attachment`

## Extension Type

Instance extension

## Registered Extension Number

64

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Bryce Hutchings, Microsoft  
Yin Li, Microsoft  
Alex Turner, Microsoft

## Overview



This extension enables the runtime to attach to app-provided `HolographicSpace` and `CoreWindow` WinRT objects when an `XrSession` is created. Applications may use this extension to create and control the `CoreWindow/App View` objects, allowing the app to subscribe to keyboard input events and react to activation event arguments. These events and data would otherwise be inaccessible if the application simply managed the app state and lifetime exclusively through the OpenXR API. This extension is only valid to use where an application can create a `CoreWindow`, such as UWP applications on the HoloLens.

The `XrHolographicWindowAttachmentMSFT` structure is defined as:

```
// Provided by XR_MSFT_holographic_window_attachment
typedef struct XrHolographicWindowAttachmentMSFT {
    XrStructureType    type;
    const void*        next;
    IUnknown*          holographicSpace;
    IUnknown*          coreWindow;
} XrHolographicWindowAttachmentMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `holographicSpace` is a pointer to a valid `Windows.Graphics.Holographic.HolographicSpace`.
- `coreWindow` is a pointer to a valid `Windows.UI.Core.CoreWindow`.

When creating a holographic window-backed `XrSession`, the application provides a pointer to an `XrHolographicWindowAttachmentMSFT` in the `next` chain of the `XrSessionCreateInfo`.

The session state of a holographic window-backed `XrSession` will only reach `XR_SESSION_STATE_VISIBLE` when the provided `CoreWindow` is made visible. If the `CoreWindow` is for a secondary app view, the application must programmatically request to make the `CoreWindow` visible (e.g. with `ApplicationViewSwitcher.TryShowAsStandaloneAsync` or `ApplicationViewSwitcher.SwitchAsync`).

The app **must** not call `xrCreateSession` while the specified `CoreWindow` thread is blocked, otherwise the call **may** deadlock.

## Valid Usage (Implicit)

- The `XR_MSFT_holographic_window_attachment` extension **must** be enabled prior to using `XrHolographicWindowAttachmentMSFT`
- `type` **must** be `XR_TYPE_HOLOGRAPHIC_WINDOW_ATTACHMENT_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `holographicSpace` **must** be a pointer to an `IUnknown` value
- `coreWindow` **must** be a pointer to an `IUnknown` value

### 12.115.1. Sample code

Following example demos the usage of holographic window attachment and use the attached `CoreWindow` to receive keyboard input, use `CoreTextEditContext` to handle text typing experience, and use `IActivatedEventArgs` to handle protocol launching arguments.

```
1 struct AppView : implements<AppView, IFrameworkView> {
2     void Initialize(CoreApplicationView const& applicationView) {
3         applicationView.Activated({this, &AppView::OnActivated});
4     }
5
6     void Load(winrt::hstring const& entryPoint) {
7     }
8
9     void Uninitialize() {
10    }
11
12    void Run() {
13        // Creating a HolographicSpace before activating the CoreWindow to make it a
        holographic window
14        CoreWindow window = CoreWindow::GetForCurrentThread();
15        HolographicSpace holographicSpace = Windows::Graphics::Holographic
        ::HolographicSpace::CreateForCoreWindow(window);
16        window.Activate();
17
18        // [xrCreateInstance, xrGetSystem, and create a graphics binding]
19
20        XrHolographicWindowAttachmentMSFT holographicWindowAttachment
        {XR_TYPE_ATTACHED_CORE_WINDOW_MSFT};
21        holographicWindowAttachment.next = &graphicsBinding;
22        holographicWindowAttachment.coreWindow = window.as<IUnknown>().get();
23        holographicWindowAttachment.holographicSpace = holographicSpace.as<IUnknown
        >().get();
24    }
```

```

25     XrSessionCreateInfo sessionCreateInfo{XR_TYPE_SESSION_CREATE_INFO};
26     sessionCreateInfo.next = &holographicWindowAttachment;
27     sessionCreateInfo.systemId = systemId;
28
29     XrSession session;
30     CHECK_XRCMD(xrCreateSession(instance, &sessionCreateInfo, &session));
31
32     while (!m_windowClosed) {
33         window.Dispatcher().ProcessEvents(CoreProcessEventsOption
34         ::ProcessAllIfPresent);
35         // [OpenXR calls: Poll events, sync actions, render, and submit frames].
36     }
37 }
38
39 void SetWindow(CoreWindow const& window) {
40     window.Closed({this, &AppView::OnWindowClosed});
41     window.KeyDown({this, &AppView::OnKeyDown});
42
43     // This sample customizes the text input pane with manual display policy and
44     email address scope.
45     windows::CoreTextServicesManager manager = windows::CoreTextServicesManager
46     ::GetForCurrentView();
47     windows::CoreTextEditContext editingContext = manager.CreateEditContext();
48     editingContext.InputPaneDisplayPolicy(windows::
49     CoreTextInputPaneDisplayPolicy::Manual);
50     editingContext.InputScope(windows::CoreTextInputScope::EmailAddress);
51 }
52
53 void OnWindowClosed(CoreWindow const& sender, CoreWindowEventArgs const& args) {
54     m_windowClosed = true;
55 }
56
57 void OnKeyDown(CoreWindow const& sender, KeyEventArgs const& args) {
58     // [Process key down]
59 }
60
61 void OnActivated(CoreApplicationView const&, IActivatedEventArgs const& args) {
62     if (args.Kind() == windows::ActivationKind::Protocol) {
63         auto eventArgs{args.as<windows::ProtocolActivatedEventArgs>()};
64         // Use the protocol activation parameters in eventArgs.Uri();
65     }
66
67     // Inspecting whether the application is launched from within holographic
68     shell or from desktop.
69     if (windows::HolographicApplicationPreview::IsHolographicActivation(args)) {
70         // App activation is targeted at the holographic shell.
71     } else {

```

```

68     // App activation is targeted at the desktop.
69     }
70
71     // NOTE: CoreWindow is activated later after the HolographicSpace has been
    created.
72     }
73
74     bool m_windowClosed{false};
75 };
76
77 struct AppViewSource : winrt::implements<AppViewSource, IFrameworkViewSource> {
78     windows::IFrameworkView CreateView() {
79         return winrt::make<AppView>();
80     }
81 };
82
83 int __stdcall wWinMain(HINSTANCE, HINSTANCE, PWSTR, int) {
84    CoreApplication::Run(make<AppViewSource>());
85 }

```

## Version History

- Revision 1, 2020-05-18 (Bryce Hutchings)
  - Initial extension description

# 12.116. XR\_MSFT\_perception\_anchor\_interop

## Name String

`XR_MSFT_perception_anchor_interop`

## Extension Type

Instance extension

## Registered Extension Number

57

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_MSFT\\_spatial\\_anchor](#)

## Last Modified Date

2020-06-16

## IP Status

No known IP claims.

## Contributors

Lachlan Ford, Microsoft  
Bryce Hutchings, Microsoft  
Yin Li, Microsoft

## Overview

This extension supports conversion between [XrSpatialAnchorMSFT](#) and [Windows.Perception.Spatial.SpatialAnchor](#). An application **can** use this extension to persist spatial anchors on the Windows device through [SpatialAnchorStore](#) or transfer spatial anchors between devices through [SpatialAnchorTransferManager](#).

The [xrCreateSpatialAnchorFromPerceptionAnchorMSFT](#) function creates a [XrSpatialAnchorMSFT](#) handle from an [IUnknown](#) pointer to [Windows.Perception.Spatial.SpatialAnchor](#).

```
// Provided by XR_MSFT_perception_anchor_interop
XrResult xrCreateSpatialAnchorFromPerceptionAnchorMSFT(
    XrSession session,
    IUnknown* perceptionAnchor,
    XrSpatialAnchorMSFT* anchor);
```

## Parameter Descriptions

- `session` is the specified [XrSession](#).
- `perceptionAnchor` is an [IUnknown](#) pointer to a [Windows.Perception.Spatial.SpatialAnchor](#) object.
- `anchor` is a pointer to [XrSpatialAnchorMSFT](#) to receive the returned anchor handle.

The input `perceptionAnchor` **must** support successful [QueryInterface](#) to [Windows.Perception.Spatial.SpatialAnchor](#), otherwise the runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#).

If the function successfully returned, the output `anchor` **must** be a valid handle. This also increments the refcount of the `perceptionAnchor` object.

When application is done with the `anchor` handle, it **can** be destroyed using `xrDestroySpatialAnchorMSFT` function. This also decrements the refcount of underlying windows perception anchor object.

### Valid Usage (Implicit)

- The `XR_MSFT_perception_anchor_interop` extension **must** be enabled prior to calling `xrCreateSpatialAnchorFromPerceptionAnchorMSFT`
- `session` **must** be a valid `XrSession` handle
- `perceptionAnchor` **must** be a pointer to an `IUnknown` value
- `anchor` **must** be a pointer to an `XrSpatialAnchorMSFT` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The `xrTryGetPerceptionAnchorFromSpatialAnchorMSFT` function converts a `XrSpatialAnchorMSFT` handle into an `IUnknown` pointer to `Windows.Perception.Spatial.SpatialAnchor`.

```
// Provided by XR_MSFT_perception_anchor_interop
XrResult xrTryGetPerceptionAnchorFromSpatialAnchorMSFT(
    XrSession session,
    XrSpatialAnchorMSFT anchor,
    IUnknown** perceptionAnchor);
```

## Parameter Descriptions

- `session` is the specified `XrSession`.
- `anchor` is a valid `XrSpatialAnchorMSFT` handle.
- `perceptionAnchor` is a valid pointer to `IUnknown` pointer to receive the output `Windows.Perception.Spatial.SpatialAnchor` object.

If the runtime can convert the `anchor` to a `Windows.Perception.Spatial.SpatialAnchor` object, this function **must** return `XR_SUCCESS`, and the output `IUnknown` in the pointer of `perceptionAnchor` **must** be not `NULL`. This also increments the refcount of the object. The application **can** then use `QueryInterface` to get the pointer for `Windows.Perception.Spatial.SpatialAnchor` object. The application **should** release the COM pointer after done with the object, or attach it to a smart COM pointer such as `winrt::com_ptr`.

If the runtime cannot convert the `anchor` to a `Windows.Perception.Spatial.SpatialAnchor` object, the function **must** return `XR_SUCCESS`, and the output `IUnknown` in the pointer of `perceptionAnchor` **must** be `NULL`.

## Valid Usage (Implicit)

- The `XR_MSFT_perception_anchor_interop` extension **must** be enabled prior to calling `xrTryGetPerceptionAnchorFromSpatialAnchorMSFT`
- `session` **must** be a valid `XrSession` handle
- `anchor` **must** be a valid `XrSpatialAnchorMSFT` handle
- `perceptionAnchor` **must** be a pointer to a pointer to an `IUnknown` value
- `anchor` **must** have been created, allocated, or retrieved from `session`

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

### New Object Types

### New Flag Types

### New Enum Constants

### New Enums

### New Structures

### New Functions

[xrCreateSpatialAnchorFromPerceptionAnchorMSFT](#)

[xrTryGetPerceptionAnchorFromSpatialAnchorMSFT](#)

### Issues

### Version History

- Revision 1, 2020-06-16 (Yin Li)
  - Initial extension proposal

## 12.117. XR\_MSFT\_scene\_marker

### Name String

`XR_MSFT_scene_marker`



**Extension Type**

Instance extension

**Registered Extension Number**

148

**Revision**

1

**Extension and Version Dependencies**

[OpenXR 1.0](#)

and

[XR\\_MSFT\\_scene\\_understanding](#)

**Contributors**

Alain Zanchetta, Microsoft

Yin Li, Microsoft

Alex Turner, Microsoft

**12.117.1. Overview**

This extension enables the application to observe the tracked markers, such as the QR Code markers in [ISO/IEC 18004:2015](#). This extension also enables future extensions to easily add new types of marker tracking.

The application **must** enable both [XR\\_MSFT\\_scene\\_marker](#) and [XR\\_MSFT\\_scene\\_understanding](#) in order to use this extension.

### Note

A typical use of this extension is:

1. Verify if marker detection is supported by calling [xrEnumerateSceneComputeFeaturesMSFT](#) and validate that the returned supported features include `XR_SCENE_COMPUTE_FEATURE_MARKER_MSFT`.
2. If supported, create an [XrSceneObserverMSFT](#) handle.
3. Pass in `XR_SCENE_COMPUTE_FEATURE_MARKER_MSFT` as requested feature when starting the scene compute by calling [xrComputeNewSceneMSFT](#) function.
4. Inspect the completion of computation by polling [xrGetSceneComputeStateMSFT](#).
5. Once compute is successfully completed, create an [XrSceneMSFT](#) handle to the result by calling [xrCreateSceneMSFT](#).
6. Get the list of detected markers using [xrGetSceneComponentsMSFT](#):
  - optionally: filter the type of the returned markers using [XrSceneMarkerTypeFilterMSFT](#).
  - optionally: retrieve additional marker properties by chaining [XrSceneMarkersMSFT](#) and/or [XrSceneMarkerQRCodesMSFT](#) to the next pointer of [XrSceneComponentsMSFT](#).
7. Get the data encoded in a marker using [xrGetSceneMarkerDecodedStringMSFT](#) or [xrGetSceneMarkerRawDataMSFT](#).
8. Locate markers using [xrLocateSceneComponentsMSFT](#).



## 12.117.2. Retrieve marker properties

The [XrSceneMarkersMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_marker
typedef struct XrSceneMarkersMSFT {
    XrStructureType    type;
    const void*        next;
    uint32_t           sceneMarkerCapacityInput;
    XrSceneMarkerMSFT* sceneMarkers;
} XrSceneMarkersMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. See also: `XrSceneComponentsMSFT`, `XrSceneMarkerQRCodesMSFT`
- `sceneMarkerCapacityInput` is a `uint32_t` indicating the capacity of elements in the `sceneMarkers` array.
- `sceneMarkers` is an array of `XrSceneMarkerMSFT` to fill with the properties of the markers.

Once the application creates an `XrSceneMSFT` after a successful scene compute, it **can** retrieve the scene markers' properties by chaining `XrSceneMarkersMSFT` structure to the next pointer of `XrSceneComponentsGetInfoMSFT` when calling `xrGetSceneComponentsMSFT`.

`xrGetSceneComponentsMSFT` follows the [two-call idiom](#) for filling the `XrSceneComponentsMSFT` structure to which an `XrSceneMarkersMSFT` structure **can** be chained.

The input `sceneMarkerCapacityInput` **must** be equal to or greater than the corresponding `XrSceneComponentsMSFT::componentCapacityInput`, otherwise the runtime **must** return `XR_ERROR_SIZE_INSUFFICIENT`.

The actual count of elements returned in the array `sceneMarkers` is consistent with the extended `XrSceneComponentsMSFT` structure and returned in `XrSceneComponentsMSFT::componentCountOutput`.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_marker` extension **must** be enabled prior to using `XrSceneMarkersMSFT`
- `type` **must** be `XR_TYPE_SCENE_MARKERS_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `sceneMarkerCapacityInput` is not `0`, `sceneMarkers` **must** be a pointer to an array of `sceneMarkerCapacityInput` `XrSceneMarkerMSFT` structures

The `XrSceneMarkerMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_marker
typedef struct XrSceneMarkerMSFT {
    XrSceneMarkerTypeMSFT    markerType;
    XrTime                    lastSeenTime;
    XrOffset2Df               center;
    XrExtent2Df               size;
} XrSceneMarkerMSFT;
```

## Member Descriptions

- **markerType** is an [XrSceneMarkerTypeMSFT](#) indicating the type of the marker.
- **lastSeenTime** is an [XrTime](#) indicating when the marker was seen last.
- **center** is an [XrOffset2Df](#) structure representing the location of the center of the axis-aligned bounding box of the marker in the XY plane of the marker's coordinate system.
- **size** is an [XrExtent2Df](#) structure representing the width and height of the axis-aligned bounding box of the marker in the XY plane of the marker's coordinate system.

The [XrSceneMarkerMSFT](#) structure is an element in the array of [XrSceneMarkersMSFT::sceneMarkers](#).

Refer to the [QR code convention](#) for an example of marker's center and size in the context of a QR code.

When the runtime updates the location or properties of an observed marker, the runtime **must** set the [XrSceneMarkerMSFT::lastSeenTime](#) to the new timestamp of the update.

When the runtime cannot observe a previously observed [XrSceneMarkerMSFT](#), the runtime **must** keep the previous **lastSeenTime** for the marker. Hence, the application **can** use the **lastSeenTime** to know how fresh the tracking information is for a given marker.

The **center** and **size** are measured in meters, relative to the [XrPosef](#) of the marker for the visual bound of the marker in XY plane, regardless of the marker type.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_marker](#) extension **must** be enabled prior to using [XrSceneMarkerMSFT](#)

The [XrSceneMarkerTypeFilterMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_marker
typedef struct XrSceneMarkerTypeFilterMSFT {
    XrStructureType      type;
    const void*          next;
    uint32_t             markerTypeCount;
    XrSceneMarkerTypeMSFT* markerTypes;
} XrSceneMarkerTypeFilterMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `markerTypeCount` is a `uint32_t` indicating the count of elements in the `markerTypes` array.
- `markerTypes` is an array of [XrSceneMarkerTypeMSFT](#) indicating the types of markers to return.

The application **can** filter the returned scene components to specific marker types by chaining [XrSceneMarkerTypeFilterMSFT](#) to the `next` pointer of [XrSceneComponentsGetInfoMSFT](#) when calling [xrGetSceneComponentsMSFT](#).

When [XrSceneMarkerTypeFilterMSFT](#) is provided to [xrGetSceneComponentsMSFT](#), the runtime **must** only return scene components that match the requested types.

The application **must** provide a non-empty array of unique `markerTypes`, i.e. the `markerTypeCount` **must** be positive and the elements in the `markerTypes` array **must** not have duplicated values. Otherwise, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE` for [xrGetSceneComponentsMSFT](#) function.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_marker` extension **must** be enabled prior to using [XrSceneMarkerTypeFilterMSFT](#)
- `type` **must** be `XR_TYPE_SCENE_MARKER_TYPE_FILTER_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `markerTypeCount` is not `0`, `markerTypes` **must** be a pointer to an array of `markerTypeCount` [XrSceneMarkerTypeMSFT](#) values

The [XrSceneMarkerTypeMSFT](#) identifies the type of a scene marker.

```
// Provided by XR_MSFT_scene_marker
typedef enum XrSceneMarkerTypeMSFT {
    XR_SCENE_MARKER_TYPE_QR_CODE_MSFT = 1,
    XR_SCENE_MARKER_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneMarkerTypeMSFT;
```

## Enumerant Descriptions

- `XR_SCENE_MARKER_TYPE_QR_CODE_MSFT` represents a marker that follows the ISO standard for QR code in [ISO/IEC 18004:2015](#).

### 12.117.3. Locate markers

Applications **can** use `xrLocateSceneComponentsMSFT` to locate an `XrSceneMarkerMSFT`.

The scene marker's locations are snapshots of the `XrSceneMSFT`, that do not change for the lifecycle of the result. To get updated tracking, the application **can** issue another `xrComputeNewSceneMSFT` and obtain a new `XrSceneMSFT`. The application **can** use the `XrSceneComponentMSFT::id` to correlate the same marker across multiple scene computes.

The pose and geometry of scene markers returned from this extension follows these general conventions:

- The marker image reside in the plane of X and Y axes.
- Z axis is perpendicular to the X and Y axes and follows the right hand rule. +Z is pointing into the marker image.
- The origin of the marker is runtime defined for the specific `XrSceneMarkerTypeMSFT`, and it typically represents the most stable and accurate point for tracking the marker. This allows the application to use the marker as a tracked point.
- In cases where the origin does not necessarily coincide with the center of the marker geometry, applications can obtain additional geometry information from the `XrSceneMarkerMSFT` structure. This information includes the center and size of the marker image in the X and Y plane.

The exact origin and geometry properties relative to the tracked marker image in physical world **must** be well defined and consistent for each `XrSceneMarkerTypeMSFT`, including the new marker types defined in future extensions.

### 12.117.4. The convention of QRCode marker location

For a marker with `XR_SCENE_MARKER_TYPE_QR_CODE_MSFT`, the origin is at the top left corner of the QR code image, where the orientation of the QR code image in the XY plane follows the convention in [ISO/IEC 18004:2015](#). The X axis of QR code pose points to the right of the marker image, and the Z axis points

inward to the marker image, as illustrated in following image.

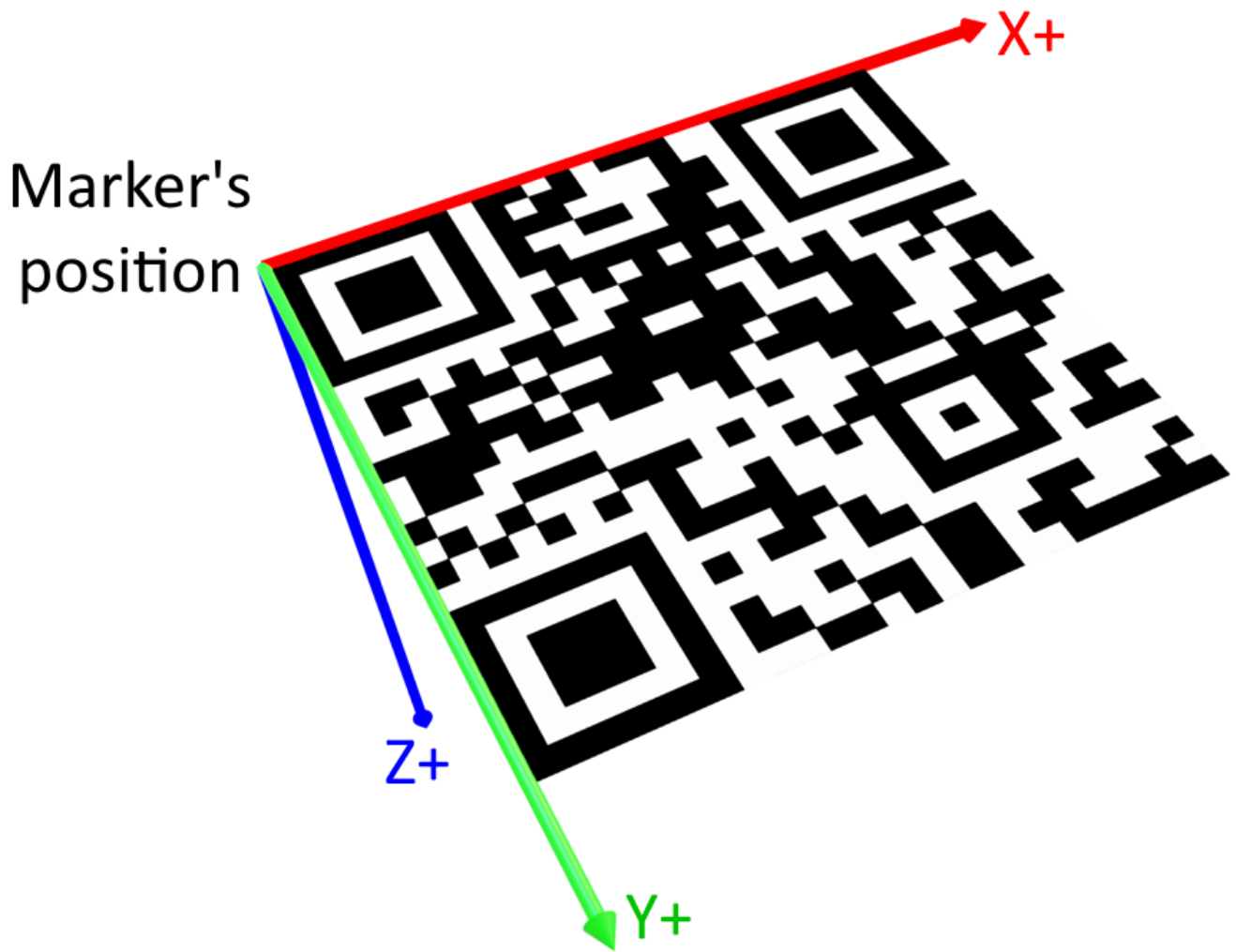


Figure 17. The pose convention of a QR code marker.

The QR Code marker's center and size are defined in the XY plane, as illustrated in following pictures.

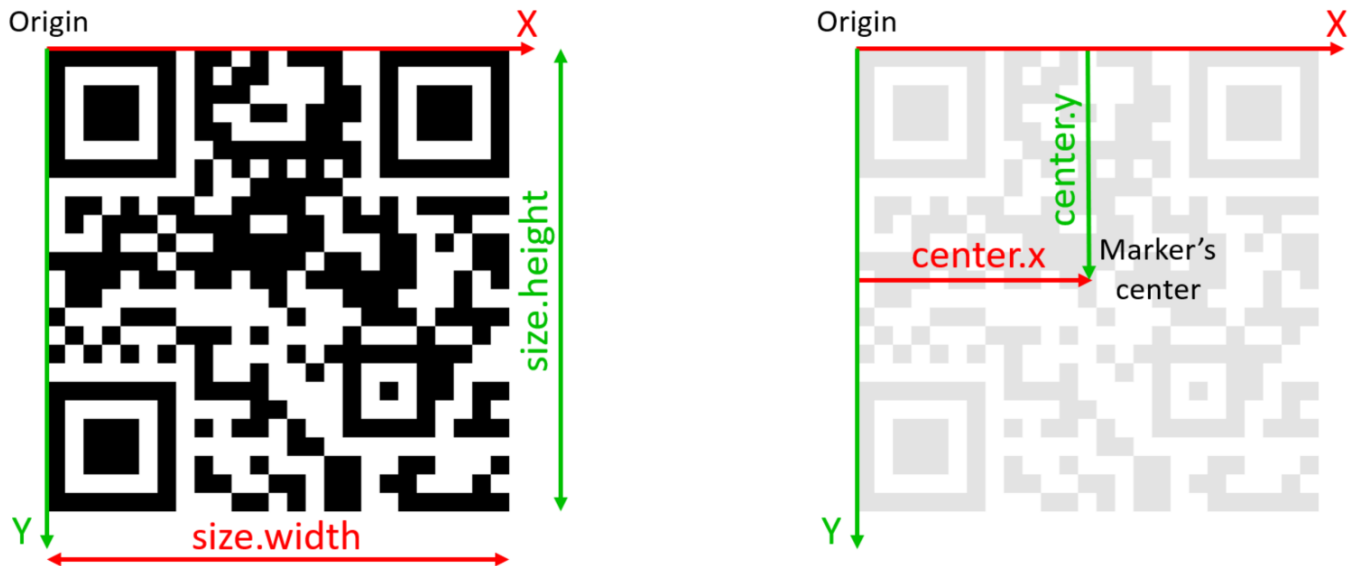


Figure 18. The center and size of QR Code marker.

### 12.117.5. Retrieving QRCode marker properties

The `XrSceneMarkerQRCodeMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_marker
typedef struct XrSceneMarkerQRCodeMSFT {
    XrStructureType      type;
    const void*         next;
    uint32_t             qrCodeCapacityInput;
    XrSceneMarkerQRCodeMSFT* qrCodes;
} XrSceneMarkerQRCodeMSFT;
```

#### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. See also: `XrSceneComponentsMSFT`, `XrSceneMarkersMSFT`
- `qrCodeCapacityInput` is a `uint32_t` indicating the count of elements in the `qrCodes` array.
- `qrCodes` is an array of `XrSceneMarkerQRCodeMSFT` for the runtime to fill with the properties of the QR Codes.

An `XrSceneMarkerQRCodeMSFT` structure can be chained to the `next` pointer of `XrSceneComponentsMSFT` when calling `xrGetSceneComponentsMSFT` function to retrieve the QR Code



specific properties through an array of [XrSceneMarkerQRCodeMSFT](#) structures.

[xrGetSceneComponentsMSFT](#) follows the [two-call idiom](#) for filling the [XrSceneComponentsMSFT](#) structure to which an [XrSceneMarkerQRCodesMSFT](#) structure **can** be chained.

The `qrCodeCapacityInput` **must** be equal to or greater than the corresponding [XrSceneComponentsMSFT::componentCapacityInput](#), otherwise the runtime **must** return the success code `XR_ERROR_SIZE_INSUFFICIENT` from [xrGetSceneComponentsMSFT](#).

The actual count of elements returned in the array `qrCodes` is consistent to the extended [XrSceneComponentsMSFT](#) structure and returned in [XrSceneComponentsMSFT::componentCountOutput](#).

### Valid Usage (Implicit)

- The `XR_MSFT_scene_marker` extension **must** be enabled prior to using [XrSceneMarkerQRCodesMSFT](#)
- `type` **must** be `XR_TYPE_SCENE_MARKER_QR_CODES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `qrCodeCapacityInput` is not `0`, `qrCodes` **must** be a pointer to an array of `qrCodeCapacityInput` [XrSceneMarkerQRCodeMSFT](#) structures

The [XrSceneMarkerQRCodeMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_marker
typedef struct XrSceneMarkerQRCodeMSFT {
    XrSceneMarkerQRCodeSymbolTypeMSFT    symbolType;
    uint8_t                               version;
} XrSceneMarkerQRCodeMSFT;
```

### Member Descriptions

- `symbolType` is an [XrSceneMarkerQRCodeSymbolTypeMSFT](#) indicating the symbol type of the QR Code.
- `version` is a `uint8_t` indicating the version of the QR Code

The [XrSceneMarkerQRCodeMSFT](#) structure contains the detailed QR Code symbol type and version according to [ISO/IEC 18004:2015](#). The `version` **must** be in the range 1 to 40 inclusively for a QR Code and 1 to 4 inclusively for a Micro QR Code.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_marker` extension **must** be enabled prior to using `XrSceneMarkerQRCodeMSFT`

```
// Provided by XR_MSFT_scene_marker
typedef enum XrSceneMarkerQRCodeSymbolTypeMSFT {
    XR_SCENE_MARKER_QR_CODE_SYMBOL_TYPE_QR_CODE_MSFT = 1,
    XR_SCENE_MARKER_QR_CODE_SYMBOL_TYPE_MICRO_QR_CODE_MSFT = 2,
    XR_SCENE_MARKER_QR_CODE_SYMBOL_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneMarkerQRCodeSymbolTypeMSFT;
```

The `XrSceneMarkerQRCodeSymbolTypeMSFT` identifies the symbol type of the QR Code.

## Enumerant Descriptions

- `XR_SCENE_MARKER_QR_CODE_SYMBOL_TYPE_QR_CODE_MSFT` if the marker is a QR Code.
- `XR_SCENE_MARKER_QR_CODE_SYMBOL_TYPE_MICRO_QR_CODE_MSFT` if the marker is a Micro QR Code.

The `xrGetSceneMarkerDecodedStringMSFT` function is defined as:

```
// Provided by XR_MSFT_scene_marker
XrResult xrGetSceneMarkerDecodedStringMSFT(
    XrSceneMSFT scene,
    const XrUuidMSFT* markerId,
    uint32_t bufferCapacityInput,
    uint32_t* bufferCountOutput,
    char* buffer);
```

## Parameter Descriptions

- `scene` is an [XrSceneMSFT](#) previously created by [xrCreateSceneMSFT](#).
- `markerId` is an [XrUuidMSFT](#) identifying the marker, returned previously from [XrSceneComponentMSFT::id](#) when calling [xrGetSceneComponentsMSFT](#).
- `bufferCapacityInput` is the capacity of the string buffer, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is a pointer to the count of characters written (including the terminating '\0'), or a pointer to the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an application-allocated buffer that will be filled with the string stored in the QR Code. It can be NULL if `bufferCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

The [xrGetSceneMarkerDecodedStringMSFT](#) function retrieves the string stored in the scene marker as an UTF-8 string, including the terminating '\0'. This function follows the [two-call idiom](#) for filling the `buffer` array.

If the stored data in the marker is not an encoded string, the runtime **must** return the success code `XR_SCENE_MARKER_DATA_NOT_STRING_MSFT`, set `bufferCountOutput` to 1, and make `buffer` an empty string.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_marker](#) extension **must** be enabled prior to calling [xrGetSceneMarkerDecodedStringMSFT](#)
- `scene` **must** be a valid [XrSceneMSFT](#) handle
- `markerId` **must** be a pointer to a valid [XrUuidMSFT](#) structure
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` char values

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING
- XR\_SCENE\_MARKER\_DATA\_NOT\_STRING\_MSFT

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_SCENE\_COMPONENT\_TYPE\_MISMATCH\_MSFT
- XR\_ERROR\_SCENE\_COMPONENT\_ID\_INVALID\_MSFT

The `xrGetSceneMarkerRawDataMSFT` function is defined as:

```
// Provided by XR_MSFT_scene_marker
XrResult xrGetSceneMarkerRawDataMSFT(
    XrSceneMSFT scene,
    const XrUuidMSFT* markerId,
    uint32_t bufferCapacityInput,
    uint32_t* bufferCountOutput,
    uint8_t* buffer);
```

## Parameter Descriptions

- `scene` is an [XrSceneMSFT](#) previously created by [xrCreateSceneMSFT](#).
- `markerId` is an [XrUuidMSFT](#) identifying the marker, and it is returned previous from [XrSceneComponentMSFT](#) when calling [xrGetSceneComponentsMSFT](#).
- `bufferCapacityInput` is the capacity of the buffer, or 0 to indicate a request to retrieve the required capacity.
- `bufferCountOutput` is a pointer to the count of bytes written, or a pointer to the required capacity in the case that `bufferCapacityInput` is insufficient.
- `buffer` is a pointer to an application-allocated buffer that will be filled with the data stored in the QR Code. It can be NULL if `bufferCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `buffer` size.

The [xrGetSceneMarkerRawDataMSFT](#) function retrieves the data stored in the scene marker.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_marker](#) extension **must** be enabled prior to calling [xrGetSceneMarkerRawDataMSFT](#)
- `scene` **must** be a valid [XrSceneMSFT](#) handle
- `markerId` **must** be a pointer to a valid [XrUuidMSFT](#) structure
- `bufferCountOutput` **must** be a pointer to a `uint32_t` value
- If `bufferCapacityInput` is not 0, `buffer` **must** be a pointer to an array of `bufferCapacityInput` `uint8_t` values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SCENE_COMPONENT_TYPE_MISMATCH_MSFT`
- `XR_ERROR_SCENE_COMPONENT_ID_INVALID_MSFT`

## New Object Types

## New Flag Types

## New Enum Constants

`XrSceneComputeFeatureMSFT` enumeration is extended with:

- `XR_SCENE_COMPUTE_FEATURE_MARKER_MSFT`

`XrSceneComponentTypeMSFT` enumeration is extended with:

- `XR_SCENE_COMPONENT_TYPE_MARKER_MSFT`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SCENE_MARKERS_MSFT`
- `XR_TYPE_SCENE_MARKER_TYPE_FILTER_MSFT`
- `XR_TYPE_SCENE_MARKER_QR_CODES_MSFT`

`XrResult` enumeration is extended with:

- `XR_SCENE_MARKER_DATA_NOT_STRING_MSFT`

## New Enums

- [XrSceneMarkerTypeMSFT](#)
- [XrSceneMarkerQRCodeSymbolTypeMSFT](#)

## New Structures

- [XrSceneMarkerMSFT](#)
- [XrSceneMarkersMSFT](#)
- [XrSceneMarkerTypeFilterMSFT](#)
- [XrSceneMarkerQRCodeMSFT](#)
- [XrSceneMarkerQRCodesMSFT](#)

## New Functions

- [xrGetSceneMarkerRawDataMSFT](#)
- [xrGetSceneMarkerDecodedStringMSFT](#)

## Version History

- Revision 1, 2023-01-11 (Alain Zanchetta)
  - Initial extension description

# 12.118. XR\_MSFT\_scene\_understanding

## Name String

[XR\\_MSFT\\_scene\\_understanding](#)

## Extension Type

Instance extension

## Registered Extension Number

98

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-05-03

## IP Status

No known IP claims.

## Contributors

Darryl Gough, Microsoft

Yin Li, Microsoft

Bryce Hutchings, Microsoft

Alex Turner, Microsoft

Simon Stachniak, Microsoft

David Fields, Microsoft

## Overview

Scene understanding provides applications with a structured, high-level representation of the planes, meshes, and objects in the user's environment, enabling the development of spatially-aware applications.

The application requests computation of a scene, receiving the list of scene components observed in the environment around the user. These scene components contain information such as:

- The type of the discovered objects (wall, floor, ceiling, or other surface type).
- The planes and their bounds that represent the object.
- The visual and collider triangle meshes that represent the object.

The application can use this information to reason about the structure and location of the environment, to place holograms on surfaces, or render clues for grounding objects.

An application typically uses this extension in the following steps:

1. Create an [XrSceneObserverMSFT](#) handle to manage the system resource of the scene understanding compute.
2. Start the scene compute by calling [xrComputeNewSceneMSFT](#) with [XrSceneBoundsMSFT](#) to specify the scan range and a list of [XrSceneComputeFeatureMSFT](#) features.
3. Inspect the completion of computation by polling [xrGetSceneComputeStateMSFT](#).
4. Once compute is completed, create an [XrSceneMSFT](#) handle to the result by calling [xrCreateSceneMSFT](#).
5. Get properties of scene components using [xrGetSceneComponentsMSFT](#).
6. Locate scene components using [xrLocateSceneComponentsMSFT](#).

## Create a scene observer handle

The [XrSceneObserverMSFT](#) handle represents the resources for computing scenes. It maintains a correlation of scene component identifiers across multiple scene computes.





#### Note

The application should destroy the [XrSceneObserverMSFT](#) handle when it is done with scene compute and scene component data to save system power consumption.

```
XR_DEFINE_HANDLE(XrSceneObserverMSFT)
```

An [XrSceneObserverMSFT](#) handle is created using [xrCreateSceneObserverMSFT](#).

```
// Provided by XR_MSFT_scene_understanding
XrResult xrCreateSceneObserverMSFT(
    XrSession session,
    const XrSceneObserverCreateInfoMSFT* createInfo,
    XrSceneObserverMSFT* sceneObserver);
```

### Parameter Descriptions

- `session` is an [XrSession](#) in which the scene observer will be active.
- `createInfo` exists for extensibility purposes, it is `NULL` or a pointer to a valid [XrSceneObserverCreateInfoMSFT](#) structure.
- `sceneObserver` is the returned [XrSceneObserverMSFT](#) handle.

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to calling [xrCreateSceneObserverMSFT](#)
- `session` **must** be a valid [XrSession](#) handle
- If `createInfo` is not `NULL`, `createInfo` **must** be a pointer to a valid [XrSceneObserverCreateInfoMSFT](#) structure
- `sceneObserver` **must** be a pointer to an [XrSceneObserverMSFT](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The `XrSceneObserverCreateInfoMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneObserverCreateInfoMSFT {
    XrStructureType    type;
    const void*        next;
} XrSceneObserverCreateInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneObserverCreateInfoMSFT`
- `type` **must** be `XR_TYPE_SCENE_OBSERVER_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `xrDestroySceneObserverMSFT` function releases the `sceneObserver` and the underlying resources.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrDestroySceneObserverMSFT(
    XrSceneObserverMSFT          sceneObserver);
```

### Parameter Descriptions

- `sceneObserver` is an `XrSceneObserverMSFT` previously created by `xrCreateSceneObserverMSFT`.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to calling `xrDestroySceneObserverMSFT`
- `sceneObserver` **must** be a valid `XrSceneObserverMSFT` handle

### Thread Safety

- Access to `sceneObserver`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## Compute a new scene and wait for completion

The `xrComputeNewSceneMSFT` function begins the compute of a new scene and the runtime **must** return quickly without waiting for the compute to complete. The application **should** use `xrGetSceneComputeStateMSFT` to inspect the compute status.

The application **can** control the compute features by passing a list of `XrSceneComputeFeatureMSFT` via `XrNewSceneComputeInfoMSFT::requestedFeatures`.

- If `XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT` is passed, but `XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT` is not passed, then:
  - The application **may** be able to read `XR_SCENE_COMPONENT_TYPE_PLANE_MSFT` and `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` scene components from the resulting `XrSceneMSFT` handle.
  - `XrScenePlaneMSFT::meshBufferId` **must** be zero to indicate that the plane scene component does not have a mesh buffer available to read.
- If `XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT` and `XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT` are passed, then:
  - the application **may** be able to read `XR_SCENE_COMPONENT_TYPE_PLANE_MSFT` and `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` scene components from the resulting `XrSceneMSFT` handle
  - `XrScenePlaneMSFT::meshBufferId` **may** contain a non-zero mesh buffer identifier to indicate that the plane scene component has a mesh buffer available to read.
- If `XR_SCENE_COMPUTE_FEATURE_VISUAL_MESH_MSFT` is passed then:
  - the application **may** be able to read `XR_SCENE_COMPONENT_TYPE_VISUAL_MESH_MSFT` and `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` scene components from the resulting `XrSceneMSFT` handle.
- If `XR_SCENE_COMPUTE_FEATURE_COLLIDER_MESH_MSFT` is passed then:
  - the application **may** be able to read `XR_SCENE_COMPONENT_TYPE_COLLIDER_MESH_MSFT` and `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` scene components from the resulting `XrSceneMSFT` handle.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrComputeNewSceneMSFT(
    XrSceneObserverMSFT          sceneObserver,
    const XrNewSceneComputeInfoMSFT* computeInfo);
```

## Parameter Descriptions

- `sceneObserver` is a handle to an [XrSceneObserverMSFT](#).
- `computeInfo` is a pointer to an [XrNewSceneComputeInfoMSFT](#) structure.

The runtime **must** return `XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT` if incompatible features were passed or no compatible features were passed.

The runtime **must** return `XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT` if `XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT` was passed but `XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT` was not passed.

The runtime **must** return `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT` if `xrComputeNewSceneMSFT` is called while the scene computation is in progress.

An application that wishes to use `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` **must** create an [XrSceneObserverMSFT](#) handle that passes neither `XR_SCENE_COMPUTE_CONSISTENCY_SNAPSHOT_COMPLETE_MSFT` nor `XR_SCENE_COMPUTE_CONSISTENCY_SNAPSHOT_INCOMPLETE_FAST_MSFT` to `xrComputeNewSceneMSFT` for the lifetime of that [XrSceneObserverMSFT](#) handle. This allows the runtime to return occlusion mesh at a different cadence than non-occlusion mesh or planes.

- The runtime **must** return `XR_ERROR_SCENE_COMPUTE_CONSISTENCY_MISMATCH_MSFT` if:
  - `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` is passed to `xrComputeNewSceneMSFT` **and**
  - a previous call to `xrComputeNewSceneMSFT` did not pass `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` for the same [XrSceneObserverMSFT](#) handle.
- The runtime **must** return `XR_ERROR_SCENE_COMPUTE_CONSISTENCY_MISMATCH_MSFT` if:
  - `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` is not passed to `xrComputeNewSceneMSFT` **and**
  - a previous call to `xrComputeNewSceneMSFT` did pass `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` for the same [XrSceneObserverMSFT](#) handle.

- The runtime **must** return `XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT` if:
  - `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` is passed to `xrComputeNewSceneMSFT` **and**
  - neither `XR_SCENE_COMPUTE_FEATURE_VISUAL_MESH_MSFT` nor `XR_SCENE_COMPUTE_FEATURE_COLLIDER_MESH_MSFT` are also passed.
- The runtime **must** return `XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT` if:
  - `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` is passed to `xrComputeNewSceneMSFT` **and**
  - at least one of `XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT`, `XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT`, `XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT`, or `XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT` are also passed.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to calling `xrComputeNewSceneMSFT`
- `sceneObserver` **must** be a valid `XrSceneObserverMSFT` handle
- `computeInfo` **must** be a pointer to a valid `XrNewSceneComputeInfoMSFT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_TIME_INVALID`
- `XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT`
- `XR_ERROR_SCENE_COMPUTE_CONSISTENCY_MISMATCH_MSFT`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT`

An [XrSceneMSFT](#) handle represents the collection of scene components that were detected during the scene computation.

```
XR_DEFINE_HANDLE(XrSceneMSFT)
```

The [XrNewSceneComputeInfoMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrNewSceneComputeInfoMSFT {
    XrStructureType                type;
    const void*                    next;
    uint32_t                       requestedFeatureCount;
    const XrSceneComputeFeatureMSFT* requestedFeatures;
    XrSceneComputeConsistencyMSFT consistency;
    XrSceneBoundsMSFT             bounds;
} XrNewSceneComputeInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `requestedFeatureCount` is the number of features.
- `requestedFeatures` is an array of [XrSceneComputeFeatureMSFT](#).
- `consistency` indicates the requested [XrSceneComputeConsistencyMSFT](#), trading off speed against the quality of the resulting scene.
- `bounds` is an [XrSceneBoundsMSFT](#) representing the culling volume. Scene components entirely outside this volume **should** culled.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrNewSceneComputeInfoMSFT](#)
- `type` **must** be `XR_TYPE_NEW_SCENE_COMPUTE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrVisualMeshComputeLodInfoMSFT](#)
- `requestedFeatures` **must** be a pointer to an array of `requestedFeatureCount` valid [XrSceneComputeFeatureMSFT](#) values
- `consistency` **must** be a valid [XrSceneComputeConsistencyMSFT](#) value
- `bounds` **must** be a valid [XrSceneBoundsMSFT](#) structure
- The `requestedFeatureCount` parameter **must** be greater than `0`

The [XrSceneComputeFeatureMSFT](#) enumeration identifies the different scene compute features that may be passed to [xrComputeNewSceneMSFT](#).



```
// Provided by XR_MSFT_scene_understanding
typedef enum XrSceneComputeFeatureMSFT {
    XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT = 1,
    XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT = 2,
    XR_SCENE_COMPUTE_FEATURE_VISUAL_MESH_MSFT = 3,
    XR_SCENE_COMPUTE_FEATURE_COLLIDER_MESH_MSFT = 4,
// Provided by XR_MSFT_scene_understanding_serialization
    XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT = 1000098000,
// Provided by XR_MSFT_scene_marker
    XR_SCENE_COMPUTE_FEATURE_MARKER_MSFT = 1000147000,
    XR_SCENE_COMPUTE_FEATURE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneComputeFeatureMSFT;
```

## Enumerant Descriptions

- `XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT` specifies that plane data for objects should be included in the resulting scene.
- `XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT` specifies that planar meshes for objects should be included in the resulting scene.
- `XR_SCENE_COMPUTE_FEATURE_VISUAL_MESH_MSFT` specifies that 3D visualization meshes for objects should be included in the resulting scene.
- `XR_SCENE_COMPUTE_FEATURE_COLLIDER_MESH_MSFT` specifies that 3D collider meshes for objects should be included in the resulting scene.

### Note

Applications wanting to use the scene for analysis, or in a physics simulation should set `consistency` to `XR_SCENE_COMPUTE_CONSISTENCY_SNAPSHOT_COMPLETE_MSFT` in order to avoid physics objects falling through the gaps and escaping the scene.



Setting `consistency` to `XR_SCENE_COMPUTE_CONSISTENCY_SNAPSHOT_INCOMPLETE_FAST_MSFT` might speed up the compute but it will result in gaps in the scene.

Setting `consistency` to `XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT` should be done when the resulting mesh will only be used to occlude virtual objects that are behind real-world surfaces. This mode will be most efficient and have the lowest-latency, but will return meshes less suitable for analysis or visualization.

The `XrSceneComputeConsistencyMSFT` enumeration identifies the different scene compute consistencies that may be passed to `xrComputeNewSceneMSFT`.

```
// Provided by XR_MSFT_scene_understanding
typedef enum XrSceneComputeConsistencyMSFT {
    XR_SCENE_COMPUTE_CONSISTENCY_SNAPSHOT_COMPLETE_MSFT = 1,
    XR_SCENE_COMPUTE_CONSISTENCY_SNAPSHOT_INCOMPLETE_FAST_MSFT = 2,
    XR_SCENE_COMPUTE_CONSISTENCY_OCCLUSION_OPTIMIZED_MSFT = 3,
    XR_SCENE_COMPUTE_CONSISTENCY_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneComputeConsistencyMSFT;
```

## Enumerant Descriptions

- **XR\_SCENE\_COMPUTE\_CONSISTENCY\_SNAPSHOT\_COMPLETE\_MSFT**. The runtime **must** return a scene that is a consistent and complete snapshot of the environment, inferring the size and shape of objects as needed where the objects were not directly observed, in order to generate a watertight representation of the scene.
- **XR\_SCENE\_COMPUTE\_CONSISTENCY\_SNAPSHOT\_INCOMPLETE\_FAST\_MSFT**. The runtime **must** return a consistent snapshot of the scene with meshes that do not overlap adjacent meshes at their edges, but **may** skip returning objects with [XrSceneObjectTypeMSFT](#) of **XR\_SCENE\_OBJECT\_TYPE\_INFERRED\_MSFT** in order to return the scene faster.
- **XR\_SCENE\_COMPUTE\_CONSISTENCY\_OCCLUSION\_OPTIMIZED\_MSFT**. The runtime **may** react to this value by computing scenes more quickly and reusing existing mesh buffer IDs more often to minimize app overhead, with potential tradeoffs such as returning meshes that are not watertight, meshes that overlap adjacent meshes at their edges to allow partial updates in the future, or other reductions in mesh quality that are less observable when mesh is used for occlusion only.

An application **can** pass one or more bounding volumes when calling [xrComputeNewSceneMSFT](#). These bounding volumes are used to determine which scene components to include in the resulting scene. Scene components that intersect one or more of the bounding volumes **should** be included, and all other scene components **should** be excluded. If an application inputs no bounding volumes, then the runtime **must** not associate any scene components with the resulting [XrSceneMSFT](#) handle.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneBoundsMSFT {
    XrSpace                space;
    XrTime                 time;
    uint32_t               sphereCount;
    const XrSceneSphereBoundMSFT* spheres;
    uint32_t               boxCount;
    const XrSceneOrientedBoxBoundMSFT* boxes;
    uint32_t               frustumCount;
    const XrSceneFrustumBoundMSFT* frustums;
} XrSceneBoundsMSFT;
```

## Member Descriptions

- `space` is a handle to the `XrSpace` in which the bounds are specified.
- `time` is the `XrTime` at which the bounds will be evaluated within `space`.
- `sphereCount` is the number of sphere bounds.
- `spheres` is an array of `XrSceneSphereBoundMSFT`.
- `boxCount` is the number of oriented box bounds.
- `boxes` is an array of `XrSceneOrientedBoxBoundMSFT`.
- `frustumCount` is the number of frustum bounds.
- `frustums` is an array of `XrSceneFrustumBoundMSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneBoundsMSFT`
- `space` **must** be a valid `XrSpace` handle
- If `sphereCount` is not `0`, `spheres` **must** be a pointer to an array of `sphereCount` `XrSceneSphereBoundMSFT` structures
- If `boxCount` is not `0`, `boxes` **must** be a pointer to an array of `boxCount` `XrSceneOrientedBoxBoundMSFT` structures
- If `frustumCount` is not `0`, `frustums` **must** be a pointer to an array of `frustumCount` `XrSceneFrustumBoundMSFT` structures

An `XrSceneSphereBoundMSFT` structure describes the center and radius of a sphere bounds.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneSphereBoundMSFT {
    XrVector3f    center;
    float        radius;
} XrSceneSphereBoundMSFT;
```

## Member Descriptions

- **center** is an [XrVector3f](#) representing the center of the sphere bound within the reference frame of the corresponding [XrSceneBoundsMSFT::space](#).
- **radius** is the finite positive radius of the sphere bound.

The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if **radius** is not a finite positive value.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneSphereBoundMSFT](#)

An [XrSceneOrientedBoxBoundMSFT](#) structure describes the pose and extents of an oriented box bounds.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneOrientedBoxBoundMSFT {
    XrPosef      pose;
    XrVector3f   extents;
} XrSceneOrientedBoxBoundMSFT;
```

## Member Descriptions

- **pose** is an [XrPosef](#) defining the center position and orientation of the oriented bounding box bound within the reference frame of the corresponding [XrSceneBoundsMSFT::space](#).
- **extents** is an [XrVector3f](#) defining the edge-to-edge length of the box along each dimension with **pose** as the center.

The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if any component of **extents** is not finite or less than or equal to zero.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneOrientedBoxBoundMSFT`

An `XrSceneFrustumBoundMSFT` structure describes the pose, field of view, and far distance of a frustum bounds.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneFrustumBoundMSFT {
    XrPosef      pose;
    XrFovf      fov;
    float       farDistance;
} XrSceneFrustumBoundMSFT;
```

## Member Descriptions

- `pose` is an `XrPosef` defining the position and orientation of the tip of the frustum bound within the reference frame of the corresponding `XrSceneBoundsMSFT::space`.
- `fov` is an `XrFovf` for the four sides of the frustum bound where `XrFovf::angleLeft` and `XrFovf::angleRight` are along the X axis and `XrFovf::angleUp` and `XrFovf::angleDown` are along the Y axis of the frustum bound space.
- `farDistance` is the positive distance of the far plane of the frustum bound along the -Z direction of the frustum bound space.

The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if `farDistance` is less than or equal to zero. The runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the `fov` angles are not between between  $-\pi/2$  and  $\pi/2$  exclusively.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneFrustumBoundMSFT`

Applications **can** request a desired visual mesh level of detail by including `XrVisualMeshComputeLodInfoMSFT` in the `XrNewSceneComputeInfoMSFT::next` chain. If `XrVisualMeshComputeLodInfoMSFT` is not included in the `XrNewSceneComputeInfoMSFT::next` chain, then `XR_MESH_COMPUTE_LOD_COARSE_MSFT` **must** be used for the visual mesh level of detail.

The [XrVisualMeshComputeLodInfoMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrVisualMeshComputeLodInfoMSFT {
    XrStructureType    type;
    const void*        next;
    XrMeshComputeLodMSFT lod;
} XrVisualMeshComputeLodInfoMSFT;
```

### Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is **NULL** or a pointer to the next structure in a structure chain.
- **lod** is the requested mesh level of detail specified by [XrMeshComputeLodMSFT](#).

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrVisualMeshComputeLodInfoMSFT](#)
- **type** **must** be [XR\\_TYPE\\_VISUAL\\_MESH\\_COMPUTE\\_LOD\\_INFO\\_MSFT](#)
- **next** **must** be **NULL** or a valid pointer to the [next structure in a structure chain](#)
- **lod** **must** be a valid [XrMeshComputeLodMSFT](#) value

The [XrMeshComputeLodMSFT](#) enumeration identifies the level of detail of visual mesh compute.

```
// Provided by XR_MSFT_scene_understanding
typedef enum XrMeshComputeLodMSFT {
    XR_MESH_COMPUTE_LOD_COARSE_MSFT = 1,
    XR_MESH_COMPUTE_LOD_MEDIUM_MSFT = 2,
    XR_MESH_COMPUTE_LOD_FINE_MSFT = 3,
    XR_MESH_COMPUTE_LOD_UNLIMITED_MSFT = 4,
    XR_MESH_COMPUTE_LOD_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrMeshComputeLodMSFT;
```

## Enumerant Descriptions

- `XR_MESH_COMPUTE_LOD_COARSE_MSFT`. Coarse mesh compute level of detail will generate roughly 100 triangles per cubic meter.
- `XR_MESH_COMPUTE_LOD_MEDIUM_MSFT`. Medium mesh compute level of detail will generate roughly 400 triangles per cubic meter.
- `XR_MESH_COMPUTE_LOD_FINE_MSFT`. Fine mesh compute level of detail will generate roughly 2000 triangles per cubic meter.
- `XR_MESH_COMPUTE_LOD_UNLIMITED_MSFT`. Unlimited mesh compute level of detail. There is no guarantee as to the number of triangles returned.

The `xrEnumerateSceneComputeFeaturesMSFT` function enumerates the supported scene compute features of the given system.

This function follows the [two-call idiom](#) for filling the `features` array.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrEnumerateSceneComputeFeaturesMSFT(
    XrInstance          instance,
    XrSystemId          systemId,
    uint32_t            featureCapacityInput,
    uint32_t*           featureCountOutput,
    XrSceneComputeFeatureMSFT* features);
```

## Parameter Descriptions

- `instance` is a handle to an `XrInstance`.
- `systemId` is the `XrSystemId` whose scene compute features will be enumerated.
- `featureCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `featureCountOutput` is a pointer to the count of scene compute features, or a pointer to the required capacity in the case that `featureCapacityInput` is insufficient.
- `features` is an array of `XrSceneComputeFeatureMSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to calling `xrEnumerateSceneComputeFeaturesMSFT`
- `instance` **must** be a valid `XrInstance` handle
- `featureCountOutput` **must** be a pointer to a `uint32_t` value
- If `featureCapacityInput` is not `0`, `features` **must** be a pointer to an array of `featureCapacityInput` `XrSceneComputeFeatureMSFT` values

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SYSTEM_INVALID`

An application **can** inspect the completion of the compute by polling `xrGetSceneComputeStateMSFT`. This function **should** typically be called once per frame per `XrSceneObserverMSFT`.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrGetSceneComputeStateMSFT(
    XrSceneObserverMSFT          sceneObserver,
    XrSceneComputeStateMSFT*    state);
```



## Parameter Descriptions

- `sceneObserver` is a handle to an `XrSceneObserverMSFT`.
- `state` is the returned `XrSceneComputeStateMSFT` value.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to calling `xrGetSceneComputeStateMSFT`
- `sceneObserver` **must** be a valid `XrSceneObserverMSFT` handle
- `state` **must** be a pointer to an `XrSceneComputeStateMSFT` value

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

`XrSceneComputeStateMSFT` identifies the different states of computing a new scene.

```
// Provided by XR_MSFT_scene_understanding
typedef enum XrSceneComputeStateMSFT {
    XR_SCENE_COMPUTE_STATE_NONE_MSFT = 0,
    XR_SCENE_COMPUTE_STATE_UPDATING_MSFT = 1,
    XR_SCENE_COMPUTE_STATE_COMPLETED_MSFT = 2,
    XR_SCENE_COMPUTE_STATE_COMPLETED_WITH_ERROR_MSFT = 3,
    XR_SCENE_COMPUTE_STATE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneComputeStateMSFT;
```

## Enumerant Descriptions

- `XR_SCENE_COMPUTE_STATE_NONE_MSFT` indicates that no scene is available, and that a scene is not being computed. The application **may** call `xrComputeNewSceneMSFT` to start computing a scene.
- `XR_SCENE_COMPUTE_STATE_UPDATING_MSFT` indicates that a new scene is being computed. Calling `xrCreateSceneMSFT` or `xrComputeNewSceneMSFT` **must** return the error `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT`.
- `XR_SCENE_COMPUTE_STATE_COMPLETED_MSFT` indicates that a new scene has completed computing. The application **may** call `xrCreateSceneMSFT` to get the results of the query or the application **may** call `xrComputeNewSceneMSFT` to start computing a new scene.
- `XR_SCENE_COMPUTE_STATE_COMPLETED_WITH_ERROR_MSFT` indicates that the new scene computation completed with an error. Calling `xrCreateSceneMSFT` **must** return a valid `XrSceneMSFT` handle but calling `xrGetSceneComponentsMSFT` with that handle **must** return zero scene components. The runtime **must** allow the application to call `xrComputeNewSceneMSFT` to try computing a scene again, even if the last call to `xrComputeNewSceneMSFT` resulted in `XR_SCENE_COMPUTE_STATE_COMPLETED_WITH_ERROR_MSFT`.

- The `xrGetSceneComputeStateMSFT` function **must** return `XR_SCENE_COMPUTE_STATE_NONE_MSFT` if it is called before `xrComputeNewSceneMSFT` is called for the first time for the given `XrSceneObserverMSFT` handle.
- After calling `xrComputeNewSceneMSFT` but before the asynchronous operation has completed, any calls to `xrGetSceneComputeStateMSFT` **should** return `XR_SCENE_COMPUTE_STATE_UPDATING_MSFT`.
- Once the asynchronous operation has completed successfully, `xrGetSceneComputeStateMSFT` **must** return `XR_SCENE_COMPUTE_STATE_COMPLETED_MSFT` until `xrComputeNewSceneMSFT` is called again.

### Create a scene handle after a new scene compute has completed

The `xrCreateSceneMSFT` functions creates an `XrSceneMSFT` handle. It can only be called after `xrGetSceneComputeStateMSFT` returns `XR_SCENE_COMPUTE_STATE_COMPLETED_MSFT` to indicate that the asynchronous operation has completed. The `XrSceneMSFT` handle manages the collection of scene components that represents the detected objects found during the query.

After an [XrSceneMSFT](#) handle is created, the handle and associated data **must** remain valid until destroyed, even after [xrCreateSceneMSFT](#) is called again to create the next scene. The runtime **must** keep alive any component data and mesh buffers relating to this historical scene until its handle is destroyed.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrCreateSceneMSFT(
    XrSceneObserverMSFT          sceneObserver,
    const XrSceneCreateInfoMSFT* createInfo,
    XrSceneMSFT*                 scene);
```

### Parameter Descriptions

- `sceneObserver` is a handle to an [XrSceneObserverMSFT](#).
- `createInfo` exists for extensibility purposes, it is `NULL` or a pointer to a valid [XrSceneCreateInfoMSFT](#) structure.
- `scene` is the returned [XrSceneMSFT](#) handle.

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to calling [xrCreateSceneMSFT](#)
- `sceneObserver` **must** be a valid [XrSceneObserverMSFT](#) handle
- If `createInfo` is not `NULL`, `createInfo` **must** be a pointer to a valid [XrSceneCreateInfoMSFT](#) structure
- `scene` **must** be a pointer to an [XrSceneMSFT](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT`

Calling `xrCreateSceneMSFT` when `xrGetSceneComputeStateMSFT` returns `XR_SCENE_COMPUTE_STATE_NONE_MSFT` or `XR_SCENE_COMPUTE_STATE_UPDATING_MSFT` **must** return the error `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT`.

The `XrSceneCreateInfoMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneCreateInfoMSFT {
    XrStructureType    type;
    const void*        next;
} XrSceneCreateInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneCreateInfoMSFT`
- `type` **must** be `XR_TYPE_SCENE_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The `xrDestroySceneMSFT` function releases the `scene` and the underlying resources.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrDestroySceneMSFT(
    XrSceneMSFT scene);
```

## Parameter Descriptions

- `scene` is an `XrSceneMSFT` previously created by `xrCreateSceneMSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to calling `xrDestroySceneMSFT`
- `scene` **must** be a valid `XrSceneMSFT` handle

## Thread Safety

- Access to `scene`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## Scene component types and Universally Unique Identifiers

Each [XrSceneMSFT](#) may contain one or more scene components. Scene components are uniquely identified by a Universally Unique Identifier, represented by [XrUuidMSFT](#). Each scene component belongs to one [XrSceneComponentTypeMSFT](#). The [XrSceneComponentTypeMSFT](#) denotes which additional properties can be read for that scene component.

- Get a list of scene objects and their properties in the scene by calling [xrGetSceneComponentsMSFT](#) with [XR\\_SCENE\\_COMPONENT\\_TYPE\\_OBJECT\\_MSFT](#) and including [XrSceneObjectsMSFT](#) in the [XrSceneComponentsMSFT::next](#) chain.
- Get the list of scene planes and their properties in the scene if [XR\\_SCENE\\_COMPUTE\\_FEATURE\\_PLANE\\_MSFT](#) was passed to [xrComputeNewSceneMSFT](#) by calling [xrGetSceneComponentsMSFT](#) with [XR\\_SCENE\\_COMPONENT\\_TYPE\\_PLANE\\_MSFT](#) and including [XrScenePlanesMSFT](#) in the [XrSceneComponentsMSFT::next](#) chain.
- Get the list of scene visual meshes and their properties in the scene if [XR\\_SCENE\\_COMPUTE\\_FEATURE\\_VISUAL\\_MESH\\_MSFT](#) was passed to [xrComputeNewSceneMSFT](#) by calling [xrGetSceneComponentsMSFT](#) with [XR\\_SCENE\\_COMPONENT\\_TYPE\\_VISUAL\\_MESH\\_MSFT](#) and including [XrSceneMeshesMSFT](#) in the [XrSceneComponentsMSFT::next](#) chain.
- Get the list of scene collider meshes and their properties in the scene if [XR\\_SCENE\\_COMPUTE\\_FEATURE\\_COLLIDER\\_MESH\\_MSFT](#) was passed to [xrComputeNewSceneMSFT](#) by calling [xrGetSceneComponentsMSFT](#) with [XR\\_SCENE\\_COMPONENT\\_TYPE\\_COLLIDER\\_MESH\\_MSFT](#) and including [XrSceneMeshesMSFT](#) in the [XrSceneComponentsMSFT::next](#) chain.

The [XrUuidMSFT](#) structure is a 128-bit UUID (Universally Unique Identifier) that follows [RFC 4122 Variant 1](#). The structure is composed of 16 octets, typically with the sizes and order of the fields defined in [RFC 4122 section 4.1.2](#). The [XrUuidMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrUuidMSFT {
    uint8_t    bytes[16];
} XrUuidMSFT;
```

### Member Descriptions

- [bytes](#) is a 128-bit Variant-1 Universally Unique Identifier.

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrUuidMSFT](#)

The [XrSceneComponentTypeMSFT](#) enumeration identifies the scene component type.

```
// Provided by XR_MSFT_scene_understanding
typedef enum XrSceneComponentTypeMSFT {
    XR_SCENE_COMPONENT_TYPE_INVALID_MSFT = -1,
    XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT = 1,
    XR_SCENE_COMPONENT_TYPE_PLANE_MSFT = 2,
    XR_SCENE_COMPONENT_TYPE_VISUAL_MESH_MSFT = 3,
    XR_SCENE_COMPONENT_TYPE_COLLIDER_MESH_MSFT = 4,
// Provided by XR_MSFT_scene_understanding_serialization
    XR_SCENE_COMPONENT_TYPE_SERIALIZED_SCENE_FRAGMENT_MSFT = 1000098000,
// Provided by XR_MSFT_scene_marker
    XR_SCENE_COMPONENT_TYPE_MARKER_MSFT = 1000147000,
    XR_SCENE_COMPONENT_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneComponentTypeMSFT;
```

## Enumerant Descriptions

- [XR\\_SCENE\\_COMPONENT\\_TYPE\\_INVALID\\_MSFT](#) indicates an invalid scene component type.
- [XR\\_SCENE\\_COMPONENT\\_TYPE\\_OBJECT\\_MSFT](#) indicates a discrete object detected in the world, such as a wall, floor, ceiling or table. Scene objects then provide their geometric representations such as planes and meshes as child scene components with the types below.
- [XR\\_SCENE\\_COMPONENT\\_TYPE\\_PLANE\\_MSFT](#) indicates a flat 2D representation of a surface in the world, such as a wall, floor, ceiling or table.
- [XR\\_SCENE\\_COMPONENT\\_TYPE\\_VISUAL\\_MESH\\_MSFT](#) indicates a visual mesh representation of an object in the world, optimized for visual quality when directly rendering a wireframe or other mesh visualization to the user. Visual mesh can also be used for rendering the silhouettes of objects. Applications can request varying levels of detail for visual meshes when calling [xrComputeNewSceneMSFT](#) using [XrVisualMeshComputeLodInfoMSFT](#).
- [XR\\_SCENE\\_COMPONENT\\_TYPE\\_COLLIDER\\_MESH\\_MSFT](#) indicates a collider mesh representation of an object in the world, optimized to maintain the silhouette of an object while reducing detail on mostly-flat surfaces. Collider mesh is useful when calculating physics collisions or when rendering silhouettes of objects for occlusion.

## Get scene components

Scene components are read from an [XrSceneMSFT](#) using [xrGetSceneComponentsMSFT](#) and passing one [XrSceneComponentTypeMSFT](#). This function follows the [two-call idiom](#) for filling multiple buffers in a struct. Different scene component types **may** have additional properties that **can** be read by chaining additional structures to [XrSceneComponentsMSFT](#). Those additional structures **must** have an array

size that is at least as large as `XrSceneComponentsMSFT::componentCapacityInput`, otherwise the runtime **must** return `XR_ERROR_SIZE_INSUFFICIENT`.

- If `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` is passed to `xrGetSceneComponentsMSFT`, then `XrSceneObjectsMSFT` may be included in the `XrSceneComponentsMSFT::next` chain.
- If `XR_SCENE_COMPONENT_TYPE_PLANE_MSFT` is passed to `xrGetSceneComponentsMSFT`, then `XrScenePlanesMSFT` may be included in the `XrSceneComponentsMSFT::next` chain.
- If `XR_SCENE_COMPONENT_TYPE_VISUAL_MESH_MSFT` or `XR_SCENE_COMPONENT_TYPE_COLLIDER_MESH_MSFT` are passed to `xrGetSceneComponentsMSFT`, then `XrSceneMeshesMSFT` may be included in the `XrSceneComponentsMSFT::next` chain.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrGetSceneComponentsMSFT(
    XrSceneMSFT scene,
    const XrSceneComponentsGetInfoMSFT* getInfo,
    XrSceneComponentsMSFT* components);
```

## Parameter Descriptions

- `scene` is an `XrSceneMSFT` previously created by `xrCreateSceneMSFT`.
- `getInfo` is a pointer to an `XrSceneComponentsGetInfoMSFT` structure.
- `components` is the `XrSceneComponentsMSFT` output structure.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to calling `xrGetSceneComponentsMSFT`
- `scene` **must** be a valid `XrSceneMSFT` handle
- `getInfo` **must** be a pointer to a valid `XrSceneComponentsGetInfoMSFT` structure
- `components` **must** be a pointer to an `XrSceneComponentsMSFT` structure



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SCENE_COMPONENT_TYPE_MISMATCH_MSFT`

An application **can** use [XrSceneComponentsGetInfoMSFT](#) to read the state of a specific component type using the [xrGetSceneComponentsMSFT](#) function. Applications can chain one or more of following extension structures to the [XrSceneComponentsGetInfoMSFT::next](#) chain to further narrow the returned components. The returned components **must** satisfy all conditions in the extension structs.

- [XrSceneComponentParentFilterInfoMSFT](#) to return only scene components that match the given parent object identifier.
- [XrSceneObjectTypesFilterInfoMSFT](#) to return only scene components that match any of the given [XrSceneObjectTypeMSFT](#) values or if a scene component does not have an [XrSceneObjectTypeMSFT](#) property then the parent's [XrSceneObjectTypeMSFT](#) property will be compared.
- [XrScenePlaneAlignmentFilterInfoMSFT](#) to return only scene components that match any of the given [XrScenePlaneAlignmentTypeMSFT](#) values.

The [XrSceneComponentsGetInfoMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentsGetInfoMSFT {
    XrStructureType      type;
    const void*          next;
    XrSceneComponentTypeMSFT componentType;
} XrSceneComponentsGetInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `componentType` is the scene component type requested.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneComponentsGetInfoMSFT](#)
- `type` **must** be `XR_TYPE_SCENE_COMPONENTS_GET_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrSceneComponentParentFilterInfoMSFT](#), [XrSceneMarkerTypeFilterMSFT](#), [XrSceneObjectTypesFilterInfoMSFT](#), [XrScenePlaneAlignmentFilterInfoMSFT](#)
- `componentType` **must** be a valid [XrSceneComponentTypeMSFT](#) value

The [XrSceneComponentsMSFT](#) structure contains an array of [XrSceneComponentMSFT](#) returning the components that satisfy the conditions in [xrGetSceneComponentsMSFT::getInfo](#). The [XrSceneComponentsMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentsMSFT {
    XrStructureType      type;
    void*                next;
    uint32_t              componentCapacityInput;
    uint32_t              componentCountOutput;
    XrSceneComponentMSFT* components;
} XrSceneComponentsMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `componentCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `componentCountOutput` is a pointer to the count of components, or a pointer to the required capacity in the case that `componentCapacityInput` is insufficient.
- `components` is an array of [XrSceneComponentMSFT](#).
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `components` size.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneComponentsMSFT](#)
- `type` **must** be `XR_TYPE_SCENE_COMPONENTS_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrSceneMarkerQRcodesMSFT](#), [XrSceneMarkersMSFT](#), [XrSceneMeshesMSFT](#), [XrSceneObjectsMSFT](#), [XrScenePlanesMSFT](#)
- If `componentCapacityInput` is not 0, `components` **must** be a pointer to an array of `componentCapacityInput` [XrSceneComponentMSFT](#) structures

The [XrSceneComponentMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentMSFT {
    XrSceneComponentTypeMSFT    componentType;
    XrUuidMSFT                  id;
    XrUuidMSFT                  parentId;
    XrTime                       updateTime;
} XrSceneComponentMSFT;
```

## Member Descriptions

- `componentType` is the [XrSceneComponentTypeMSFT](#) of the scene component.
- `id` is the [XrUuidMSFT](#) of the scene component.
- `parentId` is the [XrUuidMSFT](#) of the parent scene object. If the scene component does not have a parent, then `parentId` will be equal to zero.
- `updateTime` is the [XrTime](#) that this scene component was last updated.

The runtime **must** set `parentId` to either zero or a valid [XrUuidMSFT](#) that corresponds to a scene component of type `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` that exists in the [XrSceneMSFT](#).

### Note



The parent scene object is intended to allow scene components to be grouped. For example, the scene object for a wall might have multiple scene component children like `XR_SCENE_COMPONENT_TYPE_PLANE_MSFT`, `XR_SCENE_COMPONENT_TYPE_VISUAL_MESH_MSFT`, and `XR_SCENE_COMPONENT_TYPE_COLLIDER_MESH_MSFT`. Those child scene components would be alternative representations of the same wall.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using [XrSceneComponentMSFT](#)
- `componentType` **must** be a valid [XrSceneComponentTypeMSFT](#) value

## Get scene components using filters

The scene components that are returned by [xrGetSceneComponentsMSFT](#) **can** be filtered by chaining optional structures to [XrSceneComponentsGetInfoMSFT](#). The runtime **must** combine multiple filters with a logical AND.

The [XrSceneComponentParentFilterInfoMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentParentFilterInfoMSFT {
    XrStructureType    type;
    const void*        next;
    XrUuidMSFT         parentId;
} XrSceneComponentParentFilterInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `parentId` is the [XrUuidMSFT](#) of the parent scene component to filter by.

The runtime **must** return only scene components with matching `parentId`. If `parentId` is zero then the runtime **must** return only scene components that do not have a parent.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneComponentParentFilterInfoMSFT](#)
- `type` **must** be `XR_TYPE_SCENE_COMPONENT_PARENT_FILTER_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrSceneObjectTypesFilterInfoMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneObjectTypesFilterInfoMSFT {
    XrStructureType          type;
    const void*              next;
    uint32_t                  objectTypeCount;
    const XrSceneObjectTypeMSFT* objectTypes;
} XrSceneObjectTypesFilterInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `objectTypeCount` is a `uint32_t` describing the count of elements in the `objectTypes` array.
- `objectTypes` is an array of [XrSceneObjectTypeMSFT](#) to filter by.

The runtime **must** return only scene components that match any of the [XrSceneObjectTypeMSFT](#) in `objectTypes`. If a scene component does not have an [XrSceneObjectTypeMSFT](#) then the parent's [XrSceneObjectTypeMSFT](#) value will be used for the comparison if it exists.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneObjectTypesFilterInfoMSFT`
- `type` **must** be `XR_TYPE_SCENE_OBJECT_TYPES_FILTER_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- If `objectTypeCount` is not `0`, `objectTypes` **must** be a pointer to an array of `objectTypeCount` valid `XrSceneObjectTypeMSFT` values

The `XrScenePlaneAlignmentFilterInfoMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrScenePlaneAlignmentFilterInfoMSFT {
    XrStructureType          type;
    const void*              next;
    uint32_t                  alignmentCount;
    const XrScenePlaneAlignmentTypeMSFT* alignments;
} XrScenePlaneAlignmentFilterInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `alignmentCount` is a `uint32_t` describing the count of elements in the `alignments` array.
- `alignments` is an array of `XrScenePlaneAlignmentTypeMSFT` to filter by.

The runtime **must** return only scene components that match one of the `XrScenePlaneAlignmentTypeMSFT` values passed in `alignments`.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrScenePlaneAlignmentFilterInfoMSFT`
- `type` **must** be `XR_TYPE_SCENE_PLANE_ALIGNMENT_FILTER_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- If `alignmentCount` is not 0, `alignments` **must** be a pointer to an array of `alignmentCount` valid `XrScenePlaneAlignmentTypeMSFT` values

## Get scene objects

The runtime **must** fill out the `XrSceneObjectsMSFT` structure when included in the `XrSceneComponentsMSFT::next` chain. The `XrSceneComponentsGetInfoMSFT::componentType` **must** be `XR_SCENE_COMPONENT_TYPE_OBJECT_MSFT` when `XrSceneObjectsMSFT` is included in the next chain. If it is not, the `XR_ERROR_SCENE_COMPONENT_TYPE_MISMATCH_MSFT` error **must** be returned.

The `XrSceneObjectsMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneObjectsMSFT {
    XrStructureType    type;
    void*              next;
    uint32_t           sceneObjectCount;
    XrSceneObjectMSFT* sceneObjects;
} XrSceneObjectsMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `sceneObjectCount` is a `uint32_t` describing the count of elements in the `sceneObjects` array.
- `sceneObjects` is an array of `XrSceneObjectMSFT`.

The runtime **must** only set `XrSceneObjectMSFT::objectType` to any of the following `XrSceneObjectTypeMSFT` values:

- `XR_SCENE_OBJECT_TYPE_UNCATEGORIZED_MSFT`
- `XR_SCENE_OBJECT_TYPE_BACKGROUND_MSFT`

- `XR_SCENE_OBJECT_TYPE_WALL_MSFT`
- `XR_SCENE_OBJECT_TYPE_FLOOR_MSFT`
- `XR_SCENE_OBJECT_TYPE_CEILING_MSFT`
- `XR_SCENE_OBJECT_TYPE_PLATFORM_MSFT`
- `XR_SCENE_OBJECT_TYPE_INFERRED_MSFT`

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneObjectsMSFT`
- `type` **must** be `XR_TYPE_SCENE_OBJECTS_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `sceneObjectCount` is not `0`, `sceneObjects` **must** be a pointer to an array of `sceneObjectCount` `XrSceneObjectMSFT` structures

The `XrSceneObjectMSFT` structure represents the state of a scene object.

It is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneObjectMSFT {
    XrSceneObjectTypeMSFT    objectType;
} XrSceneObjectMSFT;
```

### Member Descriptions

- `objectType` is the type of the object specified by [XrSceneObjectTypeMSFT](#).

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneObjectMSFT`
- `objectType` **must** be a valid `XrSceneObjectTypeMSFT` value

The `XrSceneObjectTypeMSFT` enumeration identifies the different types of scene objects.



```
// Provided by XR_MSFT_scene_understanding
typedef enum XrSceneObjectTypeMSFT {
    XR_SCENE_OBJECT_TYPE_UNCATEGORIZED_MSFT = -1,
    XR_SCENE_OBJECT_TYPE_BACKGROUND_MSFT = 1,
    XR_SCENE_OBJECT_TYPE_WALL_MSFT = 2,
    XR_SCENE_OBJECT_TYPE_FLOOR_MSFT = 3,
    XR_SCENE_OBJECT_TYPE_CEILING_MSFT = 4,
    XR_SCENE_OBJECT_TYPE_PLATFORM_MSFT = 5,
    XR_SCENE_OBJECT_TYPE_INFERRED_MSFT = 6,
    XR_SCENE_OBJECT_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneObjectTypeMSFT;
```

## Enumerant Descriptions

- **XR\_SCENE\_OBJECT\_TYPE\_UNCATEGORIZED\_MSFT**. This scene object has yet to be classified and assigned a type. This should not be confused with background, as this object could be anything; the system has just not come up with a strong enough classification for it yet.
- **XR\_SCENE\_OBJECT\_TYPE\_BACKGROUND\_MSFT**. The scene object is known to be not one of the other recognized types of scene object. This class should not be confused with uncategorized where background is known not to be wall/floor/ceiling etc. while uncategorized is not yet categorized.
- **XR\_SCENE\_OBJECT\_TYPE\_WALL\_MSFT**. A physical wall. Walls are assumed to be immovable environmental structures.
- **XR\_SCENE\_OBJECT\_TYPE\_FLOOR\_MSFT**. Floors are any surfaces on which one can walk. Note: stairs are not floors. Also note, that floors assume any walkable surface and therefore there is no explicit assumption of a singular floor. Multi-level structures, ramps, etc. should all classify as floor.
- **XR\_SCENE\_OBJECT\_TYPE\_CEILING\_MSFT**. The upper surface of a room.
- **XR\_SCENE\_OBJECT\_TYPE\_PLATFORM\_MSFT**. A large flat surface on which you could place holograms. These tend to represent tables, countertops, and other large horizontal surfaces.
- **XR\_SCENE\_OBJECT\_TYPE\_INFERRED\_MSFT**. An imaginary object that was added to the scene in order to make the scene watertight and avoid gaps.

### Get scene planes

The runtime **must** fill out the [XrScenePlanesMSFT](#) structure when included in the [XrSceneComponentsMSFT::next](#) chain. The [XrSceneComponentsGetInfoMSFT::componentType](#) **must** be [XR\\_SCENE\\_COMPONENT\\_TYPE\\_PLANE\\_MSFT](#) when [XrScenePlanesMSFT](#) is included in the next chain. If it is not, the [XR\\_ERROR\\_SCENE\\_COMPONENT\\_TYPE\\_MISMATCH\\_MSFT](#) error **must** be returned.

The `XrScenePlanesMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrScenePlanesMSFT {
    XrStructureType    type;
    void*              next;
    uint32_t           scenePlaneCount;
    XrScenePlaneMSFT* scenePlanes;
} XrScenePlanesMSFT;
```

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `scenePlaneCount` is a `uint32_t` describing the count of elements in the `XrScenePlaneMSFT` array.
- `scenePlanes` is an array of `XrScenePlaneMSFT`.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrScenePlanesMSFT`
- `type` **must** be `XR_TYPE_SCENE_PLANES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `scenePlaneCount` is not `0`, `scenePlanes` **must** be a pointer to an array of `scenePlaneCount` `XrScenePlaneMSFT` structures

The `XrScenePlaneMSFT` structure represents the state of a scene plane.

It is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrScenePlaneMSFT {
    XrScenePlaneAlignmentTypeMSFT alignment;
    XrExtent2Df size;
    uint64_t meshBufferId;
    XrBool32 supportsIndicesUint16;
} XrScenePlaneMSFT;
```

## Member Descriptions

- **alignment** is the alignment type of the plane specified by [XrScenePlaneAlignmentTypeMSFT](#).
- **size** is the 2D size of the plane's extent, where [XrExtent2Df::width](#) is the width of the plane along the X axis, and [XrExtent2Df::height](#) is the height of the plane along the Y axis.
- **meshBufferId** is the [uint64\\_t](#) identifier that specifies the scene mesh buffer of this plane's triangle mesh. If **meshBufferId** is zero then this plane does not have a mesh. The triangles in a planar mesh are coplanar.
- **supportsIndicesUint16** is [XR\\_TRUE](#) if the mesh supports reading 16-bit unsigned indices.

The **size** of a plane refers to the plane's size in the x-y plane of the plane's coordinate system. A plane with a position of {0,0,0}, rotation of {0,0,0,1} (no rotation), and an extent of {1,1} refers to a 1 meter x 1 meter plane centered at {0,0,0} with its front face normal vector pointing towards the +Z direction in the plane component's space. For planes with an alignment of [XR\\_SCENE\\_PLANE\\_ALIGNMENT\\_TYPE\\_VERTICAL\\_MSFT](#), the +Y direction **must** point up away from the direction of gravity.

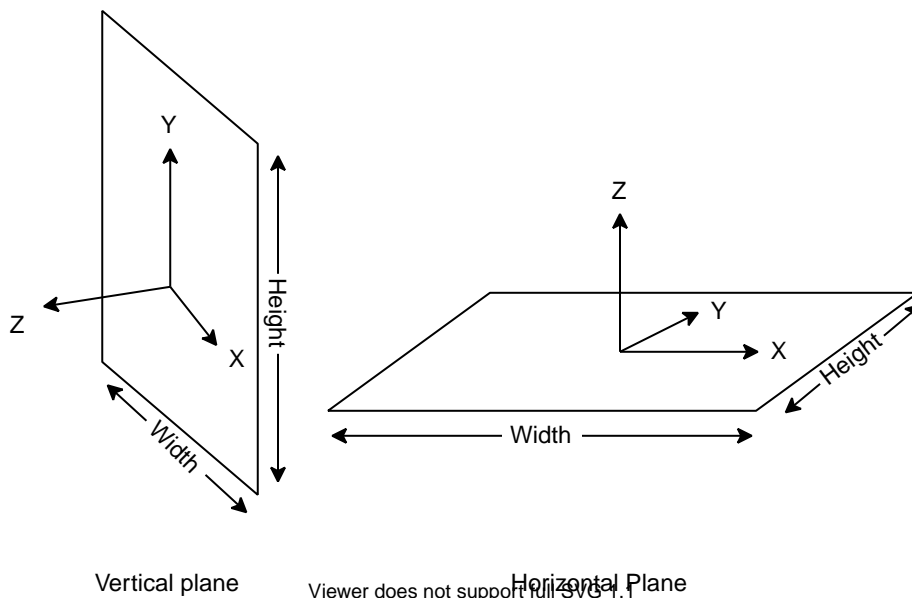


Figure 19. Scene Understanding Plane Coordinate System



### Note

OpenXR uses an X-Y plane with +Z as the plane normal but other APIs may use an X-Z plane with +Y as the plane normal. The X-Y plane can be converted to an X-Z plane by rotating  $-\pi/2$  radians around the +X axis.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrScenePlaneMSFT`
- `alignment` **must** be a valid `XrScenePlaneAlignmentTypeMSFT` value

`XrScenePlaneAlignmentTypeMSFT` identifies the different plane alignment types.

```
// Provided by XR_MSFT_scene_understanding
typedef enum XrScenePlaneAlignmentTypeMSFT {
    XR_SCENE_PLANE_ALIGNMENT_TYPE_NON_ORTHOGONAL_MSFT = 0,
    XR_SCENE_PLANE_ALIGNMENT_TYPE_HORIZONTAL_MSFT = 1,
    XR_SCENE_PLANE_ALIGNMENT_TYPE_VERTICAL_MSFT = 2,
    XR_SCENE_PLANE_ALIGNMENT_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrScenePlaneAlignmentTypeMSFT;
```

## Enumerant Descriptions

- `XR_SCENE_PLANE_ALIGNMENT_TYPE_NON_ORTHOGONAL_MSFT` means the plane's normal is not orthogonal or parallel to the gravity direction.
- `XR_SCENE_PLANE_ALIGNMENT_TYPE_HORIZONTAL_MSFT` means the plane's normal is roughly parallel to the gravity direction.
- `XR_SCENE_PLANE_ALIGNMENT_TYPE_VERTICAL_MSFT` means the plane's normal is roughly orthogonal to the gravity direction.

## Get scene mesh

The runtime **must** fill out the `XrSceneMeshesMSFT` structure when included in the `XrSceneComponentsMSFT::next` chain. The `XrSceneComponentsGetInfoMSFT::componentType` **must** be `XR_SCENE_COMPONENT_TYPE_VISUAL_MESH_MSFT` or `XR_SCENE_COMPONENT_TYPE_COLLIDER_MESH_MSFT` when `XrSceneMeshesMSFT` is included in the next chain. If it is not, the `XR_ERROR_SCENE_COMPONENT_TYPE_MISMATCH_MSFT` error **must** be returned.

The `XrSceneMeshesMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshesMSFT {
    XrStructureType    type;
    void*              next;
    uint32_t           sceneMeshCount;
    XrSceneMeshMSFT*  sceneMeshes;
} XrSceneMeshesMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `sceneMeshCount` is a `uint32_t` describing the count of elements in the `sceneMeshes` array.
- `sceneMeshes` is an array of `XrSceneMeshMSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneMeshesMSFT`
- `type` **must** be `XR_TYPE_SCENE_MESHES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- If `sceneMeshCount` is not `0`, `sceneMeshes` **must** be a pointer to an array of `sceneMeshCount` `XrSceneMeshMSFT` structures

The `XrSceneMeshMSFT` structure represents the state of a scene component's mesh.

It is defined as:

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshMSFT {
    uint64_t    meshBufferId;
    XrBool32    supportsIndicesUint16;
} XrSceneMeshMSFT;
```

## Member Descriptions

- `meshBufferId` is the `uint64_t` identifier that specifies the scene mesh buffer. If `meshBufferId` is zero then this scene component does not have mesh data of corresponding `XrSceneComponentTypeMSFT` in `xrGetSceneComponentsMSFT::getInfo`.
- `supportsIndicesUint16` is `XR_TRUE` if the mesh supports reading 16-bit unsigned indices.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneMeshMSFT`

### Read scene mesh buffer

The `xrGetSceneMeshBuffersMSFT` function retrieves the scene mesh vertex buffer and index buffer for the given scene mesh buffer identifier.

#### Note

Applications may use the scene mesh buffer identifier as a key to cache the vertices and indices of a mesh for reuse within an `XrSceneMSFT` or across multiple `XrSceneMSFT` for the same `XrSession`.



Applications can avoid unnecessarily calling `xrGetSceneMeshBuffersMSFT` for a scene component if `XrSceneComponentMSFT::updateTime` is equal to the `XrSceneComponentMSFT::updateTime` value in the previous `XrSceneMSFT`. A scene component is uniquely identified by `XrUuidMSFT`.

This function follows the two-call idiom for filling multiple buffers in a struct.

The `xrGetSceneMeshBuffersMSFT` function is defined as:

```
// Provided by XR_MSFT_scene_understanding
XrResult xrGetSceneMeshBuffersMSFT(
    XrSceneMSFT scene,
    const XrSceneMeshBuffersGetInfoMSFT* getInfo,
    XrSceneMeshBuffersMSFT* buffers);
```

## Parameter Descriptions

- `scene` is an [XrSceneMSFT](#) previously created by [xrCreateSceneMSFT](#).
- `getInfo` is a pointer to an [XrSceneMeshBuffersGetInfoMSFT](#) structure.
- `buffers` is a pointer to an [XrSceneMeshBuffersMSFT](#) structure for reading a scene mesh buffer.

Applications **can** request the vertex buffer of the mesh by including [XrSceneMeshVertexBufferMSFT](#) in the [XrSceneMeshBuffersMSFT::next](#) chain. Runtimes **must** support requesting a 32-bit index buffer and **may** support requesting a 16-bit index buffer. Applications **can** request a 32-bit index buffer by including [XrSceneMeshIndicesUint32MSFT](#) in the [XrSceneMeshBuffersMSFT::next](#) chain. Applications **can** request a 16-bit index buffer by including [XrSceneMeshIndicesUint16MSFT](#) in the [XrSceneMeshBuffersMSFT::next](#) chain. If the runtime for the given scene mesh buffer does not support requesting a 16-bit index buffer then [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) **must** be returned. The runtime **must** support reading a 16-bit index buffer for the given scene mesh buffer if [XrScenePlaneMSFT:supportsIndicesUint16](#) or [XrSceneMeshMSFT:supportsIndicesUint16](#) are [XR\\_TRUE](#) for the scene component that contained that scene mesh buffer identifier.

The runtime **must** return [XR\\_ERROR\\_SCENE\\_MESH\\_BUFFER\\_ID\\_INVALID\\_MSFT](#) if none of the scene components in the given [XrSceneMSFT](#) contain [XrSceneMeshBuffersGetInfoMSFT::meshBufferId](#). The runtime **must** return [XR\\_ERROR\\_SCENE\\_MESH\\_BUFFER\\_ID\\_INVALID\\_MSFT](#) if [XrSceneMeshBuffersGetInfoMSFT::meshBufferId](#) is zero. The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if both [XrSceneMeshIndicesUint32MSFT](#) and [XrSceneMeshIndicesUint16MSFT](#) are included in the [XrSceneMeshBuffersMSFT::next](#) chain. The runtime **must** return [XR\\_ERROR\\_VALIDATION\\_FAILURE](#) if the [XrSceneMeshBuffersMSFT::next](#) does not contain at least one of [XrSceneMeshVertexBufferMSFT](#), [XrSceneMeshIndicesUint32MSFT](#) or [XrSceneMeshIndicesUint16MSFT](#).

The runtime **must** return the same vertices and indices for a given scene mesh buffer identifier and [XrSession](#). A runtime **may** return zero vertices and indices if the underlying mesh data is no longer available.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to calling [xrGetSceneMeshBuffersMSFT](#)
- `scene` **must** be a valid [XrSceneMSFT](#) handle
- `getInfo` **must** be a pointer to a valid [XrSceneMeshBuffersGetInfoMSFT](#) structure
- `buffers` **must** be a pointer to an [XrSceneMeshBuffersMSFT](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SCENE_MESH_BUFFER_ID_INVALID_MSFT`
- `XR_ERROR_SCENE_COMPONENT_ID_INVALID_MSFT`

`XrSceneMeshBuffersGetInfoMSFT` is an input structure for the `xrGetSceneMeshBuffersMSFT` function.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshBuffersGetInfoMSFT {
    XrStructureType    type;
    const void*       next;
    uint64_t          meshBufferId;
} XrSceneMeshBuffersGetInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `meshBufferId` is the `uint64_t` identifier that specifies the scene mesh buffer to read.



## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneMeshBuffersGetInfoMSFT](#)
- **type** **must** be [XR\\_TYPE\\_SCENE\\_MESH\\_BUFFERS\\_GET\\_INFO\\_MSFT](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

[XrSceneMeshBuffersMSFT](#) is an input/output structure for reading scene mesh buffers.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshBuffersMSFT {
    XrStructureType    type;
    void*              next;
} XrSceneMeshBuffersMSFT;
```

## Member Descriptions

- **type** is the [XrStructureType](#) of this structure.
- **next** is [NULL](#) or a pointer to the next structure in a structure chain.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneMeshBuffersMSFT](#)
- **type** **must** be [XR\\_TYPE\\_SCENE\\_MESH\\_BUFFERS\\_MSFT](#)
- **next** **must** be [NULL](#) or a valid pointer to the [next structure in a structure chain](#)

[XrSceneMeshVertexBufferMSFT](#) is an input/output structure for reading scene mesh buffer vertices.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshVertexBufferMSFT {
    XrStructureType    type;
    void*              next;
    uint32_t           vertexCapacityInput;
    uint32_t           vertexCountOutput;
    XrVector3f*        vertices;
} XrSceneMeshVertexBufferMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `vertexCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `vertexCountOutput` is the count of vertices, or the required capacity in the case that `vertexCapacityInput` is insufficient.
- `vertices` is an array of [XrVector3f](#) filled in by the runtime returns the position of vertices in the mesh component's space.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `vertices` size.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneMeshVertexBufferMSFT](#)
- `type` **must** be `XR_TYPE_SCENE_MESH_VERTEX_BUFFER_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `vertexCapacityInput` is not 0, `vertices` **must** be a pointer to an array of `vertexCapacityInput` [XrVector3f](#) structures

[XrSceneMeshIndicesUint32MSFT](#) is an input/output structure for reading 32-bit indices from a scene mesh buffer.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshIndicesUint32MSFT {
    XrStructureType    type;
    void*              next;
    uint32_t           indexCapacityInput;
    uint32_t           indexCountOutput;
    uint32_t*          indices;
} XrSceneMeshIndicesUint32MSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `indexCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `indexCountOutput` is the count of indices, or the required capacity in the case that `indexCapacityInput` is insufficient.
- `indices` is an array of triangle indices filled in by the runtime, specifying the indices of the scene mesh buffer in the vertices array. The triangle indices **must** be returned in counter-clockwise order and three indices denote one triangle.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `indices` size.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneMeshIndicesUint32MSFT](#)
- `type` **must** be `XR_TYPE_SCENE_MESH_INDICES_UINT32_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `indexCapacityInput` is not 0, `indices` **must** be a pointer to an array of `indexCapacityInput` `uint32_t` values

[XrSceneMeshIndicesUint16MSFT](#) is an input/output structure for reading 16-bit indices from a scene mesh buffer.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneMeshIndicesUint16MSFT {
    XrStructureType    type;
    void*              next;
    uint32_t           indexCapacityInput;
    uint32_t           indexCountOutput;
    uint16_t*          indices;
} XrSceneMeshIndicesUint16MSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `indexCapacityInput` is the capacity of the array, or 0 to indicate a request to retrieve the required capacity.
- `indexCountOutput` is a pointer to the count of indices, or a pointer to the required capacity in the case that `indexCapacityInput` is insufficient.
- `indices` is an array of triangle indices filled in by the runtime, specifying the indices of the scene mesh buffer in the vertices array. The triangle indices **must** be returned in counter-clockwise order and three indices denote one triangle.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `indices` size.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to using [XrSceneMeshIndicesUint16MSFT](#)
- `type` **must** be `XR_TYPE_SCENE_MESH_INDICES_UINT16_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `indexCapacityInput` is not 0, `indices` **must** be a pointer to an array of `indexCapacityInput` `uint16_t` values

## Locate scene objects

The [xrLocateSceneComponentsMSFT](#) function locates an array of scene components to a base space at a given time.

```
// Provided by XR_MSFT_scene_understanding
XrResult xrLocateSceneComponentsMSFT(
    XrSceneMSFT scene,
    const XrSceneComponentsLocateInfoMSFT* locateInfo,
    XrSceneComponentLocationsMSFT* locations);
```

## Parameter Descriptions

- `scene` is a handle to an [XrSceneMSFT](#).
- `locateInfo` is a pointer to [XrSceneComponentsLocateInfoMSFT](#) describing information to locate scene components.
- `locations` is a pointer to [XrSceneComponentLocationsMSFT](#) receiving the returned scene component locations.

The runtime **must** return `XR_ERROR_SIZE_INSUFFICIENT` if [XrSceneComponentLocationsMSFT::locationCount](#) is less than [XrSceneComponentsLocateInfoMSFT::componentIdCount](#).

### Note



Similar to [xrLocateSpace](#), apps should call [xrLocateSceneComponentsMSFT](#) each frame because the location returned by [xrLocateSceneComponentsMSFT](#) in later frames may change over time as the target space or the scene components may refine their locations.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding](#) extension **must** be enabled prior to calling [xrLocateSceneComponentsMSFT](#)
- `scene` **must** be a valid [XrSceneMSFT](#) handle
- `locateInfo` **must** be a pointer to a valid [XrSceneComponentsLocateInfoMSFT](#) structure
- `locations` **must** be a pointer to an [XrSceneComponentLocationsMSFT](#) structure

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_TIME\_INVALID

The [XrSceneComponentsLocateInfoMSFT](#) structure describes the information to locate scene components.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentsLocateInfoMSFT {
    XrStructureType    type;
    const void*        next;
    XrSpace             baseSpace;
    XrTime              time;
    uint32_t           componentIdCount;
    const XrUuidMSFT*  componentIds;
} XrSceneComponentsLocateInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `baseSpace` is an `XrSpace` within which the scene components will be located.
- `time` is an `XrTime` at which to locate the scene components.
- `componentIdCount` is a `uint32_t` describing the count of elements in the `componentIds` array.
- `componentIds` is an array of `XrUuidMSFT` identifiers for the scene components to location.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneComponentsLocateInfoMSFT`
- `type` **must** be `XR_TYPE_SCENE_COMPONENTS_LOCATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `baseSpace` **must** be a valid `XrSpace` handle
- If `componentIdCount` is not `0`, `componentIds` **must** be a pointer to an array of `componentIdCount` `XrUuidMSFT` structures

The `XrSceneComponentLocationsMSFT` structure returns scene component locations.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentLocationsMSFT {
    XrStructureType      type;
    void*                next;
    uint32_t             locationCount;
    XrSceneComponentLocationMSFT* locations;
} XrSceneComponentLocationsMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `locationCount` is a `uint32_t` describing the count of elements in the `locations` array.
- `locations` is an array of `XrSceneComponentLocationMSFT` scene component locations.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneComponentLocationsMSFT`
- `type` **must** be `XR_TYPE_SCENE_COMPONENT_LOCATIONS_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- If `locationCount` is not `0`, `locations` **must** be a pointer to an array of `locationCount` `XrSceneComponentLocationMSFT` structures

The `XrSceneComponentLocationMSFT` structure describes the position and orientation of a scene component to space `XrSceneComponentsLocateInfoMSFT::baseSpace` at time `XrSceneComponentsLocateInfoMSFT::time`. If the scene component identified by `XrUuidMSFT` is not found, `flags` should be empty.

```
// Provided by XR_MSFT_scene_understanding
typedef struct XrSceneComponentLocationMSFT {
    XrSpaceLocationFlags    flags;
    XrPosef                 pose;
} XrSceneComponentLocationMSFT;
```

## Member Descriptions

- `flags` is a bitfield, with bit masks defined in `XrSpaceLocationFlagBits`, to indicate which members contain valid data.
- `pose` is an `XrPosef` defining the position and orientation of the scene component within the reference frame of the corresponding `XrSceneComponentsLocateInfoMSFT::baseSpace`.



## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding` extension **must** be enabled prior to using `XrSceneComponentLocationMSFT`
- `flags` **must** be `0` or a valid combination of `XrSpaceLocationFlagBits` values

### New Object Types

- `XrSceneObserverMSFT`
- `XrSceneMSFT`

### New Flag Types

### New Enum Constants

`XrObjectType` enumeration is extended with:

- `XR_OBJECT_TYPE_SCENE_OBSERVER_MSFT`
- `XR_OBJECT_TYPE_SCENE_MSFT`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SCENE_OBSERVER_CREATE_INFO_MSFT`
- `XR_TYPE_SCENE_CREATE_INFO_MSFT`
- `XR_TYPE_NEW_SCENE_COMPUTE_INFO_MSFT`
- `XR_TYPE_VISUAL_MESH_COMPUTE_LOD_INFO_MSFT`
- `XR_TYPE_SCENE_COMPONENTS_MSFT`
- `XR_TYPE_SCENE_COMPONENTS_GET_INFO_MSFT`
- `XR_TYPE_SCENE_COMPONENT_LOCATIONS_MSFT`
- `XR_TYPE_SCENE_COMPONENTS_LOCATE_INFO_MSFT`
- `XR_TYPE_SCENE_OBJECTS_MSFT`
- `XR_TYPE_SCENE_COMPONENT_PARENT_FILTER_INFO_MSFT`
- `XR_TYPE_SCENE_OBJECT_TYPES_FILTER_INFO_MSFT`
- `XR_TYPE_SCENE_PLANES_MSFT`
- `XR_TYPE_SCENE_PLANE_ALIGNMENT_FILTER_INFO_MSFT`
- `XR_TYPE_SCENE_MESHES_MSFT`
- `XR_TYPE_SCENE_MESH_BUFFERS_GET_INFO_MSFT`
- `XR_TYPE_SCENE_MESH_BUFFERS_MSFT`

[XrResult](#) enumeration is extended with:

- [XR\\_ERROR\\_COMPUTE\\_NEW\\_SCENE\\_NOT\\_COMPLETED\\_MSFT](#)
- [XR\\_ERROR\\_SCENE\\_COMPONENT\\_ID\\_INVALID\\_MSFT](#)
- [XR\\_ERROR\\_SCENE\\_COMPONENT\\_TYPE\\_MISMATCH\\_MSFT](#)
- [XR\\_ERROR\\_SCENE\\_MESH\\_BUFFER\\_ID\\_INVALID\\_MSFT](#)
- [XR\\_ERROR\\_SCENE\\_COMPUTE\\_FEATURE\\_INCOMPATIBLE\\_MSFT](#)
- [XR\\_ERROR\\_SCENE\\_COMPUTE\\_CONSISTENCY\\_MISMATCH\\_MSFT](#)

### **New Enums**

- [XrSceneComputeFeatureMSFT](#)
- [XrSceneComputeConsistencyMSFT](#)
- [XrSceneObjectTypeMSFT](#)
- [XrScenePlaneAlignmentTypeMSFT](#)
- [XrSceneComputeStateMSFT](#)
- [XrSceneComponentTypeMSFT](#)
- [XrMeshComputeLodMSFT](#)

### **New Structures**

- [XrSceneObserverCreateInfoMSFT](#)
- [XrSceneCreateInfoMSFT](#)
- [XrNewSceneComputeInfoMSFT](#)
- [XrUuidMSFT](#)
- [XrSceneObserverCreateInfoMSFT](#)
- [XrSceneCreateInfoMSFT](#)
- [XrNewSceneComputeInfoMSFT](#)
- [XrVisualMeshComputeLodInfoMSFT](#)
- [XrSceneSphereBoundMSFT](#)
- [XrSceneOrientedBoxBoundMSFT](#)
- [XrSceneFrustumBoundMSFT](#)
- [XrSceneBoundsMSFT](#)
- [XrSceneComponentMSFT](#)
- [XrSceneComponentsMSFT](#)
- [XrSceneComponentsGetInfoMSFT](#)

- [XrSceneComponentLocationMSFT](#)
- [XrSceneComponentLocationsMSFT](#)
- [XrSceneComponentsLocateInfoMSFT](#)
- [XrSceneObjectMSFT](#)
- [XrSceneObjectsMSFT](#)
- [XrSceneComponentParentFilterInfoMSFT](#)
- [XrSceneObjectTypesFilterInfoMSFT](#)
- [XrScenePlaneMSFT](#)
- [XrScenePlanesMSFT](#)
- [XrScenePlaneAlignmentFilterInfoMSFT](#)
- [XrSceneMeshMSFT](#)
- [XrSceneMeshesMSFT](#)
- [XrSceneMeshBuffersGetInfoMSFT](#)
- [XrSceneMeshBuffersMSFT](#)

### **New Functions**

- [xrCreateSceneObserverMSFT](#)
- [xrDestroySceneObserverMSFT](#)
- [xrCreateSceneMSFT](#)
- [xrDestroySceneMSFT](#)
- [xrComputeNewSceneMSFT](#)
- [xrGetSceneComponentsMSFT](#)
- [xrLocateSceneComponentsMSFT](#)
- [xrGetSceneMeshBuffersMSFT](#)

### **Issues**

### **Version History**

- Revision 1, 2021-05-03 (Darryl Gough)
  - Initial extension description
- Revision 2, 2022-06-29 (Darryl Gough)
  - Fix missing error codes

# 12.119. XR\_MSFT\_scene\_understanding\_serialization

## Name String

`XR_MSFT_scene_understanding_serialization`

## Extension Type

Instance extension

## Registered Extension Number

99

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_MSFT\\_scene\\_understanding](#)

## Last Modified Date

2021-05-03

## IP Status

No known IP claims.

## Contributors

Darryl Gough, Microsoft

Yin Li, Microsoft

Bryce Hutchings, Microsoft

Alex Turner, Microsoft

Simon Stachniak, Microsoft

David Fields, Microsoft

## Overview

This extension extends the scene understanding extension and enables scenes to be serialized or deserialized. It enables computing a new scene into a serialized binary stream and it enables deserializing a binary stream into an [XrSceneMSFT](#) handle.

## Serialize a scene

This extension adds `XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT` to [XrSceneComputeFeatureMSFT](#), which **can** be passed to [xrComputeNewSceneMSFT](#) plus one or more of `XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT`, `XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT`, `XR_SCENE_COMPUTE_FEATURE_VISUAL_MESH_MSFT` or `XR_SCENE_COMPUTE_FEATURE_COLLIDER_MESH_MSFT` to inform

the runtime that it **should** compute a serialized binary representation of the scene. If `XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT` is the only `XrSceneComputeFeatureMSFT` passed to `xrComputeNewSceneMSFT` then `XR_ERROR_SCENE_COMPUTE_FEATURE_INCOMPATIBLE_MSFT` **must** be returned.

If an `XrSceneMSFT` was created using `XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT` then `XR_SCENE_COMPONENT_TYPE_SERIALIZED_SCENE_FRAGMENT_MSFT` can be passed to the `xrGetSceneComponentsMSFT` function to read the list of serialized scene fragment `XrUuidMSFT` values from `XrSceneComponentMSFT::id`. The `XrUuidMSFT` of a scene fragment can be passed to `xrGetSerializedSceneFragmentDataMSFT` to read the binary data of the given scene fragment.

The application **can** call the `xrGetSerializedSceneFragmentDataMSFT` function to read the binary data of a serialized scene fragment from the `XrSceneMSFT` handle. This function follows the two-call idiom for filling the buffer.

The `xrGetSerializedSceneFragmentDataMSFT` function is defined as:

```
// Provided by XR_MSFT_scene_understanding_serialization
XrResult xrGetSerializedSceneFragmentDataMSFT(
    XrSceneMSFT scene,
    const XrSerializedSceneFragmentDataGetInfoMSFT* getInfo,
    uint32_t countInput,
    uint32_t* readOutput,
    uint8_t* buffer);
```

### Parameter Descriptions

- `scene` is the `XrSceneMSFT` handle to read from.
- `getInfo` is a pointer to an `XrSerializedSceneFragmentDataGetInfoMSFT` structure.
- `countInput` is the number of bytes that should be read.
- `readOutput` is the number of bytes read.
- `buffer` is a pointer to the buffer where the data should be copied.

The runtime **must** return `XR_ERROR_SCENE_COMPONENT_ID_INVALID_MSFT` if the given scene fragment `XrUuidMSFT` was not found.

## Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding_serialization` extension **must** be enabled prior to calling `xrGetSerializedSceneFragmentDataMSFT`
- `scene` **must** be a valid `XrSceneMSFT` handle
- `getInfo` **must** be a pointer to a valid `XrSerializedSceneFragmentDataGetInfoMSFT` structure
- `readOutput` **must** be a pointer to a `uint32_t` value
- If `countInput` is not `0`, `buffer` **must** be a pointer to an array of `countInput` `uint8_t` values

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`
- `XR_ERROR_SCENE_COMPONENT_ID_INVALID_MSFT`

The `XrSerializedSceneFragmentDataGetInfoMSFT` structure is defined as:

```
// Provided by XR_MSFT_scene_understanding_serialization
typedef struct XrSerializedSceneFragmentDataGetInfoMSFT {
    XrStructureType    type;
    const void*       next;
    XrUuidMSFT        sceneFragmentId;
} XrSerializedSceneFragmentDataGetInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `sceneFragmentId` is the [XrUuidMSFT](#) of the serialized scene fragment that was previously read from [xrGetSceneComponentsMSFT](#) with `XR_SCENE_COMPONENT_TYPE_SERIALIZED_SCENE_FRAGMENT_MSFT`.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_scene\\_understanding\\_serialization](#) extension **must** be enabled prior to using [XrSerializedSceneFragmentDataGetInfoMSFT](#)
- `type` **must** be `XR_TYPE_SERIALIZED_SCENE_FRAGMENT_DATA_GET_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## Deserialize a scene

This extension enables an application to deserialize the binary representation of a scene that was previously serialized.

For a given [XrSceneObserverMSFT](#) handle, instead of calling [xrComputeNewSceneMSFT](#), which computes the scene from the system's sensors, the application **can** use [xrDeserializeSceneMSFT](#) to produce a scene from the given binary scene fragment data.

The [xrDeserializeSceneMSFT](#) function is defined as:

```
// Provided by XR_MSFT_scene_understanding_serialization
XrResult xrDeserializeSceneMSFT(
    XrSceneObserverMSFT          sceneObserver,
    const XrSceneDeserializeInfoMSFT* deserializeInfo);
```

## Parameter Descriptions

- `sceneObserver` is a handle to an [XrSceneObserverMSFT](#) previously created with [xrCreateSceneObserverMSFT](#).
- `deserializeInfo` is a pointer to an [XrSceneDeserializeInfoMSFT](#) structure.

The `xrDeserializeSceneMSFT` function begins deserializing a list of serialized scene fragments. The runtime **must** return quickly without waiting for the deserialization to complete. The application **should** use `xrGetSceneComputeStateMSFT` to inspect the completeness of the deserialization.

The runtime **must** return `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT` if `xrDeserializeSceneMSFT` is called while the scene computation is in progress.

The `xrGetSceneComputeStateMSFT` function **must** return `XR_SCENE_COMPUTE_STATE_UPDATING_MSFT` while the deserialization is in progress, and `XR_SCENE_COMPUTE_STATE_COMPLETED_MSFT` when the deserialization has completed successfully. If the runtime fails to deserialize the binary stream, `xrGetSceneComputeStateMSFT` **must** return `XR_SCENE_COMPUTE_STATE_COMPLETED_WITH_ERROR_MSFT` to indicate that the deserialization has completed but an error occurred.

When `xrGetSceneComputeStateMSFT` returns `XR_SCENE_COMPUTE_STATE_COMPLETED_MSFT`, the application **may** call `xrCreateSceneMSFT` to create the `XrSceneMSFT` handle. If `xrCreateSceneMSFT` is called while `xrGetSceneComputeStateMSFT` returns `XR_SCENE_COMPUTE_STATE_COMPLETED_WITH_ERROR_MSFT`, a valid `XrSceneMSFT` handle **must** be returned, but that handle **must** contain zero scene components.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding_serialization` extension **must** be enabled prior to calling `xrDeserializeSceneMSFT`
- `sceneObserver` **must** be a valid `XrSceneObserverMSFT` handle
- `deserializeInfo` **must** be a pointer to a valid `XrSceneDeserializeInfoMSFT` structure



## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_COMPUTE_NEW_SCENE_NOT_COMPLETED_MSFT`

`XrSceneDeserializeInfoMSFT` is an input structure that describes the array of serialized scene fragments that will be deserialized by the `xrDeserializeSceneMSFT` function.

```
// Provided by XR_MSFT_scene_understanding_serialization
typedef struct XrSceneDeserializeInfoMSFT {
    XrStructureType          type;
    const void*             next;
    uint32_t                 fragmentCount;
    const XrDeserializeSceneFragmentMSFT* fragments;
} XrSceneDeserializeInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `fragmentCount` is the count of `XrDeserializeSceneFragmentMSFT` structures in the `fragments` array.
- `fragments` is an array of `XrDeserializeSceneFragmentMSFT`.

If the scene fragments are not in the same order as returned by `xrGetSceneComponentsMSFT` or the runtime failed to deserialize the binary data then `xrGetSceneComputeStateMSFT` **must** return `XR_SCENE_COMPUTE_STATE_COMPLETED_WITH_ERROR_MSFT`.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding_serialization` extension **must** be enabled prior to using `XrSceneDeserializeInfoMSFT`
- `type` **must** be `XR_TYPE_SCENE_DESERIALIZE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- If `fragmentCount` is not `0`, `fragments` **must** be a pointer to an array of `fragmentCount` valid `XrDeserializeSceneFragmentMSFT` structures

The `XrDeserializeSceneFragmentMSFT` structure represents a single fragment of a binary stream to be deserialized. It is defined as:

```
// Provided by XR_MSFT_scene_understanding_serialization
typedef struct XrDeserializeSceneFragmentMSFT {
    uint32_t      bufferSize;
    const uint8_t* buffer;
} XrDeserializeSceneFragmentMSFT;
```

### Member Descriptions

- `bufferSize` is the size of the `buffer` array.
- `buffer` is an array of `uint_8` data for the scene fragment to be deserialized.

### Valid Usage (Implicit)

- The `XR_MSFT_scene_understanding_serialization` extension **must** be enabled prior to using `XrDeserializeSceneFragmentMSFT`
- If `bufferSize` is not `0`, `buffer` **must** be a pointer to an array of `bufferSize` `uint8_t` values

## New Object Types

## New Flag Types

## New Enum Constants

[XrSceneComponentTypeMSFT](#) enumeration is extended with:

- [XR\\_SCENE\\_COMPONENT\\_TYPE\\_SERIALIZED\\_SCENE\\_FRAGMENT\\_MSFT](#)

[XrSceneComputeFeatureMSFT](#) enumeration is extended with:

- [XR\\_SCENE\\_COMPUTE\\_FEATURE\\_SERIALIZE\\_SCENE\\_MSFT](#)

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SERIALIZED\\_SCENE\\_FRAGMENT\\_DATA\\_GET\\_INFO\\_MSFT](#)
- [XR\\_TYPE\\_SCENE\\_DESERIALIZE\\_INFO\\_MSFT](#)

### **New Enums**

### **New Structures**

- [XrSerializedSceneFragmentDataGetInfoMSFT](#)
- [XrSceneDeserializeInfoMSFT](#)
- [XrDeserializeSceneFragmentMSFT](#)

### **New Functions**

- [xrGetSerializedSceneFragmentDataMSFT](#)
- [xrDeserializeSceneMSFT](#)

### **Issues**

### **Version History**

- Revision 1, 2021-05-03 (Darryl Gough)
  - Initial extension description
- Revision 2, 2022-06-29 (Darryl Gough)
  - Fix missing error codes

## **12.120. XR\_MSFT\_secondary\_view\_configuration**

### **Name String**

[XR\\_MSFT\\_secondary\\_view\\_configuration](#)

### **Extension Type**

Instance extension

## Registered Extension Number

54

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2020-05-02

## IP Status

No known IP claims.

## Contributors

Yin Li, Microsoft

Zonglin Wu, Microsoft

Alex Turner, Microsoft

### 12.120.1. Overview

This extension allows an application to enable support for one or more **secondary view configurations**. A secondary view configuration is a well-known set of views that the runtime can make active while a session is running. In a frame where a secondary view configuration is active, the application's single frame loop should additionally render into those active secondary views, sharing the frame waiting logic and update loop with the primary view configuration for that running session.

A proper secondary view configuration support includes following steps:

1. When calling `xrCreateInstance`, enable the `XR_MSFT_secondary_view_configuration` extension and the extension defines a concrete secondary view configuration type, for example, `XR_MSFT_first_person_observer`.
2. Inspect supported secondary view configurations using the `xrEnumerateViewConfigurations` function.
3. Enable supported secondary view configurations using the `xrBeginSession` function with an `XrSecondaryViewConfigurationSessionBeginInfoMSFT` chained extension structure.
4. Inspect if an enabled secondary view configuration is activated by the system or the user using the `xrWaitFrame` function with an `XrSecondaryViewConfigurationFrameStateMSFT` chained extension structure.
5. When a secondary view configuration is changed to active, get the latest view configuration properties using the `xrGetViewConfigurationProperties` and `xrEnumerateViewConfigurationViews` functions.

6. Create the swapchain images for the active secondary view configuration using the [xrCreateSwapchain](#) function with an [XrSecondaryViewConfigurationSwapchainCreateInfoMSFT](#) chained extension structure using [recommendedImageRectWidth](#) and [recommendedImageRectHeight](#) in the corresponding [XrViewConfigurationView](#) structure returned from [xrEnumerateViewConfigurationViews](#).
7. Locate the secondary view configuration views using the [xrLocateViews](#) function with the active secondary view configuration type.
8. Submit the composition layers using the swapchain images for an active secondary view configuration using the [xrEndFrame](#) function with the [XrSecondaryViewConfigurationFrameEndInfoMSFT](#) chained extension structure.

### 12.120.2. Enumerate supported secondary view configurations

The first step is for the application to inspect if a runtime supports certain secondary view configurations. The app uses the existing API [xrEnumerateViewConfigurations](#) for this.

For example, when the [XR\\_MSFT\\_first\\_person\\_observer](#) extension is enabled, the application will enumerate a view configuration of type [XR\\_VIEW\\_CONFIGURATION\\_TYPE\\_SECONDARY\\_MONO\\_FIRST\\_PERSON\\_OBSERVER\\_MSFT](#), and can use this secondary view configuration type in later functions.

### 12.120.3. Secondary view configuration properties

The application can inspect the properties of a secondary view configuration through the existing [xrGetViewConfigurationProperties](#), [xrEnumerateViewConfigurationViews](#) and [xrEnumerateEnvironmentBlendModes](#) functions using a supported secondary view configuration type.

The runtime **may** change the recommended properties, such as recommended image width or height, when the secondary view configuration becomes active. The application **should** use the latest recommended width and height when creating swapchain images and related resources for the active secondary view configuration.

When an application creates swapchain images for a secondary view configuration, it **can** chain a [XrSecondaryViewConfigurationSwapchainCreateInfoMSFT](#) structure to [XrSwapchainCreateInfo](#) when calling [xrCreateSwapchain](#). This hints to the runtime that the created swapchain image will be submitted to the given secondary view configuration, allowing the runtime to make optimizations for such usage when there is opportunity.

```
// Provided by XR_MSFT_secondary_view_configuration
typedef struct XrSecondaryViewConfigurationSwapchainCreateInfoMSFT {
    XrStructureType          type;
    const void*              next;
    XrViewConfigurationType viewConfigurationType;
} XrSecondaryViewConfigurationSwapchainCreateInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewConfigurationType` is the secondary view configuration type the application is intending to use this swapchain for.

If this structure is not present in the [XrSwapchainCreateInfo](#) next chain when calling [xrCreateSwapchain](#), the runtime **should** optimize the created swapchain for the primary view configuration of the session.

If the application submits a swapchain image created with one view configuration type to a composition layer for another view configuration, the runtime **may** need to copy the resource across view configurations. However, the runtime **must** correctly compose the image regardless which view configuration type was hinted when swapchain image was created.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_secondary\\_view\\_configuration](#) extension **must** be enabled prior to using [XrSecondaryViewConfigurationSwapchainCreateInfoMSFT](#)
- `type` **must** be `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_SWAPCHAIN_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `viewConfigurationType` **must** be a valid [XrViewConfigurationType](#) value

### 12.120.4. Enable secondary view configuration

The application indicates to the runtime which secondary view configurations it can support by chaining an [XrSecondaryViewConfigurationSessionBeginInfoMSFT](#) structure to the [XrSessionBeginInfo::next](#) pointer when calling [xrBeginSession](#).

The [XrSecondaryViewConfigurationSessionBeginInfoMSFT](#) structure is used by the application to indicate the list of secondary [XrViewConfigurationType](#) to enable for this session.

It is defined as:

```
// Provided by XR_MSFT_secondary_view_configuration
typedef struct XrSecondaryViewConfigurationSessionBeginInfoMSFT {
    XrStructureType          type;
    const void*              next;
    uint32_t                 viewConfigurationCount;
    const XrViewConfigurationType* enabledViewConfigurationTypes;
} XrSecondaryViewConfigurationSessionBeginInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewConfigurationCount` is the number of elements in `enabledViewConfigurationTypes`
- `enabledViewConfigurationTypes` is an array of enabled secondary view configuration types that application supports.

If there are any duplicated view configuration types in the array of `enabledViewConfigurationTypes`, the runtime **must** return error `XR_ERROR_VALIDATION_FAILURE`.

If there are any primary view configuration types in the array of `enabledViewConfigurationTypes`, the runtime **must** return error `XR_ERROR_VALIDATION_FAILURE`.

If there are any secondary view configuration types not returned by [xrEnumerateViewConfigurations](#) in the array of `enabledViewConfigurationTypes`, the runtime **must** return error `XR_ERROR_VIEW_CONFIGURATION_TYPE_UNSUPPORTED`.

## Valid Usage (Implicit)

- The `XR_MSFT_secondary_view_configuration` extension **must** be enabled prior to using `XrSecondaryViewConfigurationSessionBeginInfoMSFT`
- `type` **must** be `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_SESSION_BEGIN_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `enabledViewConfigurationTypes` **must** be a pointer to an array of `viewConfigurationCount` valid [XrViewConfigurationType](#) values
- The `viewConfigurationCount` parameter **must** be greater than 0

## 12.120.5. Per-frame active view configurations

The runtime then tells the application at each [xrWaitFrame](#) function call which of the enabled secondary view configurations are active for that frame. When extension structure [XrSecondaryViewConfigurationFrameStateMSFT](#) is chained to the [XrFrameState::next](#) pointer, the runtime writes into this structure the state of each enabled secondary view configuration.

The [XrSecondaryViewConfigurationFrameStateMSFT](#) structure returns whether the enabled view configurations are active or inactive.

It is defined as as:

```
// Provided by XR_MSFT_secondary_view_configuration
typedef struct XrSecondaryViewConfigurationFrameStateMSFT {
    XrStructureType          type;
    void*                    next;
    uint32_t                 viewConfigurationCount;
    XrSecondaryViewConfigurationStateMSFT* viewConfigurationStates;
} XrSecondaryViewConfigurationFrameStateMSFT;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewConfigurationCount` is the number of elements in `viewConfigurationStates`.
- `viewConfigurationStates` is an array of [XrSecondaryViewConfigurationStateMSFT](#) structures.

The array size `viewConfigurationCount` in the [XrSecondaryViewConfigurationFrameStateMSFT](#) structure **must** be the same as the array size enabled through [XrSecondaryViewConfigurationSessionBeginInfoMSFT](#) when calling [xrBeginSession](#) earlier, otherwise the runtime **must** return error `XR_ERROR_VALIDATION_FAILURE`.



## Valid Usage (Implicit)

- The `XR_MSFT_secondary_view_configuration` extension **must** be enabled prior to using `XrSecondaryViewConfigurationFrameStateMSFT`
- `type` **must** be `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_FRAME_STATE_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `viewConfigurationStates` **must** be a pointer to an array of `viewConfigurationCount` `XrSecondaryViewConfigurationStateMSFT` structures
- The `viewConfigurationCount` parameter **must** be greater than 0

The `XrSecondaryViewConfigurationStateMSFT` structure returns the state of an enabled secondary view configurations.

```
// Provided by XR_MSFT_secondary_view_configuration
typedef struct XrSecondaryViewConfigurationStateMSFT {
    XrStructureType      type;
    void*                next;
    XrViewConfigurationType viewConfigurationType;
    XrBool32             active;
} XrSecondaryViewConfigurationStateMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewConfigurationType` is an `XrViewConfigurationType` that represents the returned state.
- `active` is an `XrBool32` returns whether the secondary view configuration is active and displaying frames to users.

When a secondary view configuration becomes active, the application **should** render its secondary views as soon as possible, by getting their view transforms and FOV using `xrLocateViews` and then submitting composition layers to `xrEndFrame` through the `XrSecondaryViewConfigurationFrameEndInfoMSFT` extension structure. When a secondary view configuration changes from inactive to active, the runtime **may** change `XrViewConfigurationView` of the given view configuration such as the recommended image width or height. An application **should** query for latest `XrViewConfigurationView` through `xrEnumerateViewConfigurationViews` function for

the secondary view configuration and consider recreating swapchain images if necessary. The runtime **must** not change the [XrViewConfigurationView](#), including recommended image width and height of a secondary view configuration when **active** remains true until the secondary view configuration deactivated or the session has ended.

If necessary, the application **can** take longer than a frame duration to prepare by calling [xrEndFrame](#) without submitting layers for that secondary view configuration until ready. The runtime **should** delay the underlying scenario managed by the secondary view configuration until the application begins submitting frames with layers for that configuration. The active secondary view configuration composed output is undefined if the application stops submitting frames with layers for a secondary view configuration while **active** remains true.

When the runtime intends to conclude a secondary view configuration, for example when user stops video capture, the runtime makes the view configuration inactive by setting the corresponding **active** in the [XrSecondaryViewConfigurationStateMSFT](#) structure to false.

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_secondary\\_view\\_configuration](#) extension **must** be enabled prior to using [XrSecondaryViewConfigurationStateMSFT](#)
- **type** **must** be [XR\\_TYPE\\_SECONDARY\\_VIEW\\_CONFIGURATION\\_STATE\\_MSFT](#)
- **next** **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- **viewConfigurationType** **must** be a valid [XrViewConfigurationType](#) value

## 12.120.6. Locate and inspect view states of secondary view configurations

When the application calls [xrLocateViews](#), it **can** use [XrViewLocateInfo::viewConfigurationType](#) field to query the view locations and projections for any enabled [XrViewConfigurationType](#) for the running session.

The runtime **must** return [XR\\_ERROR\\_VIEW\\_CONFIGURATION\\_TYPE\\_UNSUPPORTED](#) from [xrLocateViews](#) if the specified [XrViewConfigurationType](#) is not enabled for the running session using [XrSecondaryViewConfigurationSessionBeginInfoMSFT](#) when calling [xrBeginSession](#).

If the view configuration is supported but not active, as indicated in [XrSecondaryViewConfigurationFrameStateMSFT](#), [xrLocateViews](#) will successfully return, but the resulting [XrViewState](#) **may** have [XR\\_VIEW\\_STATE\\_ORIENTATION\\_TRACKED\\_BIT](#) and [XR\\_VIEW\\_STATE\\_ORIENTATION\\_TRACKED\\_BIT](#) unset.

## 12.120.7. Submit composition layers to secondary view configurations

The application **should** submit layers each frame for all active secondary view configurations using the [xrEndFrame](#) function, by chaining the [XrSecondaryViewConfigurationFrameEndInfoMSFT](#)

structure to the next pointer of [XrFrameEndInfo](#) structure.

The [XrSecondaryViewConfigurationFrameEndInfoMSFT](#) structure is defined as as:

```
// Provided by XR_MSFT_secondary_view_configuration
typedef struct XrSecondaryViewConfigurationFrameEndInfoMSFT {
    XrStructureType                type;
    const void*                    next;
    uint32_t                       viewConfigurationCount;
    const XrSecondaryViewConfigurationLayerInfoMSFT* viewConfigurationLayersInfo;
} XrSecondaryViewConfigurationFrameEndInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewConfigurationCount` is the number of elements in `viewConfigurationLayersInfo`.
- `viewConfigurationLayersInfo` is an array of [XrSecondaryViewConfigurationLayerInfoMSFT](#), containing composition layers to be submitted for the specified active view configuration.

The view configuration type in each [XrSecondaryViewConfigurationLayerInfoMSFT](#) must be one of the view configurations enabled when calling [xrBeginSession](#) in [XrSecondaryViewConfigurationSessionBeginInfoMSFT](#), or else the runtime **must** return error `XR_ERROR_SECONDARY_VIEW_CONFIGURATION_TYPE_NOT_ENABLED_MSFT`.

The view configuration type in each [XrSecondaryViewConfigurationLayerInfoMSFT](#) must not be the primary view configuration in this session, or else the runtime **must** return error `XR_ERROR_LAYER_INVALID`. The primary view configuration layers continue to be submitted through [XrFrameEndInfo](#) directly.

If the view configuration is not active, as indicated in [XrSecondaryViewConfigurationFrameStateMSFT](#), the composition layers submitted to this view configuration **may** be ignored by the runtime. Applications **should** avoid rendering into secondary views when the view configuration is inactive.

## Valid Usage (Implicit)

- The `XR_MSFT_secondary_view_configuration` extension **must** be enabled prior to using `XrSecondaryViewConfigurationFrameEndInfoMSFT`
- `type` **must** be `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_FRAME_END_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `viewConfigurationLayersInfo` **must** be a pointer to an array of `viewConfigurationCount` valid `XrSecondaryViewConfigurationLayerInfoMSFT` structures
- The `viewConfigurationCount` parameter **must** be greater than 0

The application should submit an `XrSecondaryViewConfigurationLayerInfoMSFT` in `XrSecondaryViewConfigurationFrameEndInfoMSFT` for each active secondary view configuration type when calling `xrEndFrame`.

The `XrSecondaryViewConfigurationLayerInfoMSFT` structure is defined as as:

```
// Provided by XR_MSFT_secondary_view_configuration
typedef struct XrSecondaryViewConfigurationLayerInfoMSFT {
    XrStructureType                type;
    const void*                    next;
    XrViewConfigurationType        viewConfigurationType;
    XrEnvironmentBlendMode         environmentBlendMode;
    uint32_t                        layerCount;
    const XrCompositionLayerBaseHeader* const* layers;
} XrSecondaryViewConfigurationLayerInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `viewConfigurationType` is [XrViewConfigurationType](#) to which the composition layers will be displayed.
- `environmentBlendMode` is the [XrEnvironmentBlendMode](#) value representing the desired [environment blend mode](#) for this view configuration.
- `layerCount` is the number of composition layers in this frame for the secondary view configuration type. The maximum supported layer count is identified by [XrSystemGraphicsProperties::maxLayerCount](#). If `layerCount` is greater than the maximum supported layer count then `XR_ERROR_LAYER_LIMIT_EXCEEDED` is returned.
- `layers` is a pointer to an array of [XrCompositionLayerBaseHeader](#) pointers.

This structure is similar to the [XrFrameEndInfo](#) structure, with an extra [XrViewConfigurationType](#) field to specify the view configuration for which the submitted layers will be rendered.

The application **should** render its content for both the primary and secondary view configurations using the same [XrFrameState::predictedDisplayTime](#) reported by [xrWaitFrame](#). The runtime **must** treat both the primary views and secondary views as being submitted for the same [XrViewLocateInfo::displayTime](#) specified in the call to [xrEndFrame](#).

For layers such as quad layers whose content is identical across view configurations, the application **can** submit the same [XrCompositionLayerBaseHeader](#) structures to multiple view configurations in the same [xrEndFrame](#) function call.

For each frame, the application **should** only render and submit layers for the secondary view configurations that were active that frame, as indicated in the [XrSecondaryViewConfigurationFrameStateMSFT](#) filled in for that frame's [xrWaitFrame](#) call. The runtime **must** ignore composition layers submitted for an inactive view configuration.

## Valid Usage (Implicit)

- The `XR_MSFT_secondary_view_configuration` extension **must** be enabled prior to using `XrSecondaryViewConfigurationLayerInfoMSFT`
- `type` **must** be `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_LAYER_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain
- `viewConfigurationType` **must** be a valid `XrViewConfigurationType` value
- `environmentBlendMode` **must** be a valid `XrEnvironmentBlendMode` value
- `layers` **must** be a pointer to an array of `layerCount` valid `XrCompositionLayerBaseHeader`-based structures. See also: `XrCompositionLayerCubeKHR`, `XrCompositionLayerCylinderKHR`, `XrCompositionLayerEquirect2KHR`, `XrCompositionLayerEquirectKHR`, `XrCompositionLayerPassthroughHTC`, `XrCompositionLayerProjection`, `XrCompositionLayerQuad`
- The `layerCount` parameter **must** be greater than `0`

## New Object Types

## New Flag Types

## New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_SESSION_BEGIN_INFO_MSFT`
- `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_STATE_MSFT`
- `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_FRAME_STATE_MSFT`
- `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_FRAME_END_INFO_MSFT`
- `XR_TYPE_SECONDARY_VIEW_CONFIGURATION_LAYER_INFO_MSFT`
- `XR_ERROR_SECONDARY_VIEW_CONFIGURATION_TYPE_NOT_ENABLED_MSFT`

## New Enums

## New Structures

- `XrSecondaryViewConfigurationSessionBeginInfoMSFT`
- `XrSecondaryViewConfigurationStateMSFT`
- `XrSecondaryViewConfigurationFrameStateMSFT`
- `XrSecondaryViewConfigurationFrameEndInfoMSFT`
- `XrSecondaryViewConfigurationLayerInfoMSFT`

## New Functions

## Issues

## Version History

- Revision 1, 2019-07-30 (Yin Li)
  - Initial extension description

# 12.121. XR\_MSFT\_spatial\_anchor

## Name String

`XR_MSFT_spatial_anchor`

## Extension Type

Instance extension

## Registered Extension Number

40

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Overview

This extension allows an application to create a **spatial anchor**, an arbitrary freespace point in the user's physical environment that will then be tracked by the runtime. The runtime **should** then adjust the position and orientation of that anchor's origin over time as needed, independently of all other spaces and anchors, to ensure that it maintains its original mapping to the real world.

```
XR_DEFINE_HANDLE(XrSpatialAnchorMSFT)
```

Spatial anchors are often used in combination with an `UNBOUNDED_MSFT` reference space. `UNBOUNDED_MSFT` reference spaces adjust their origin as necessary to keep the viewer's coordinates relative to the space's origin stable. Such adjustments maintain the visual stability of content currently near the viewer, but may cause content placed far from the viewer to drift in its alignment to the real world by the time the user moves close again. By creating an `XrSpatialAnchorMSFT` where a piece of content is placed and then always rendering that content relative to its anchor's space, an application can ensure that each piece of content stays at a fixed location in the environment.

The `xrCreateSpatialAnchorMSFT` function is defined as:

```
// Provided by XR_MSFT_spatial_anchor
XrResult xrCreateSpatialAnchorMSFT(
    XrSession session,
    const XrSpatialAnchorCreateInfoMSFT* createInfo,
    XrSpatialAnchorMSFT* anchor);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession`.
- `createInfo` is a pointer to an `XrSpatialAnchorCreateInfoMSFT` structure containing information about how to create the anchor.
- `anchor` is a pointer to a handle in which the created `XrSpatialAnchorMSFT` is returned.

Creates an `XrSpatialAnchorMSFT` handle representing a spatial anchor that will track a fixed location in the physical world over time. That real-world location is specified by the position and orientation of the specified `XrSpatialAnchorCreateInfoMSFT::pose` within `XrSpatialAnchorCreateInfoMSFT::space` at `XrSpatialAnchorCreateInfoMSFT::time`.

The runtime **must** avoid long blocking operations such as networking or disk operations for `xrCreateSpatialAnchorMSFT` function. The application **may** safely use this function in UI thread. Though, the created anchor handle **may** not be ready immediately for certain operations yet. For example, the corresponding anchor space **may** not return valid location, or its location **may** not be successfully saved in anchor store.

If `XrSpatialAnchorCreateInfoMSFT::space` cannot be located relative to the environment at the moment of the call to `xrCreateSpatialAnchorMSFT`, the runtime **must** return `XR_ERROR_CREATE_SPATIAL_ANCHOR_FAILED_MSFT`.

After the anchor is created, the runtime **should** then adjust its position and orientation over time relative to other spaces so as to maintain maximum alignment to its original real-world location, even if that changes the anchor's relationship to the original `XrSpatialAnchorCreateInfoMSFT::space` used to initialize it.



## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor` extension **must** be enabled prior to calling `xrCreateSpatialAnchorMSFT`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrSpatialAnchorCreateInfoMSFT` structure
- `anchor` **must** be a pointer to an `XrSpatialAnchorMSFT` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_TIME_INVALID`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_CREATE_SPATIAL_ANCHOR_FAILED_MSFT`

The `XrSpatialAnchorCreateInfoMSFT` structure is defined as:

```
typedef struct XrSpatialAnchorCreateInfoMSFT {  
    XrStructureType    type;  
    const void*        next;  
    XrSpace             space;  
    XrPosef            pose;  
    XrTime             time;  
} XrSpatialAnchorCreateInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `space` is a handle to the [XrSpace](#) in which `pose` is specified.
- `pose` is the [XrPosef](#) within `space` at `time` that specifies the point in the real world used to initialize the new anchor.
- `time` is the [XrTime](#) at which `pose` will be evaluated within `space`.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_anchor](#) extension **must** be enabled prior to using [XrSpatialAnchorCreateInfoMSFT](#)
- `type` **must** be `XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `space` **must** be a valid [XrSpace](#) handle

The [xrCreateSpatialAnchorSpaceMSFT](#) function is defined as:

```
// Provided by XR_MSFT_spatial_anchor
XrResult xrCreateSpatialAnchorSpaceMSFT(
    XrSession session,
    const XrSpatialAnchorSpaceCreateInfoMSFT* createInfo,
    XrSpace* space);
```

## Parameter Descriptions

- `session` is a handle to an [XrSession](#).
- `createInfo` is a pointer to an [XrSpatialAnchorSpaceCreateInfoMSFT](#) structure containing information about how to create the anchor.
- `space` is a pointer to a handle in which the created [XrSpace](#) is returned.

Creates an [XrSpace](#) handle based on a spatial anchor. Application **can** provide an [XrPosef](#) to define the position and orientation of the new space's origin relative to the anchor's natural origin.

Multiple [XrSpace](#) handles may exist for a given [XrSpatialAnchorMSFT](#) simultaneously, up to some limit imposed by the runtime. The [XrSpace](#) handle must be eventually freed via the [xrDestroySpace](#) function or by destroying the parent [XrSpatialAnchorMSFT](#) handle.

### Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor` extension **must** be enabled prior to calling [xrCreateSpatialAnchorSpaceMSFT](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrSpatialAnchorSpaceCreateInfoMSFT](#) structure
- `space` **must** be a pointer to an [XrSpace](#) handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`

The [XrSpatialAnchorSpaceCreateInfoMSFT](#) structure is defined as:

```
typedef struct XrSpatialAnchorSpaceCreateInfoMSFT {
    XrStructureType      type;
    const void*          next;
    XrSpatialAnchorMSFT anchor;
    XrPosef              poseInAnchorSpace;
} XrSpatialAnchorSpaceCreateInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `anchor` is a handle to an [XrSpatialAnchorMSFT](#) previously created with [xrCreateSpatialAnchorMSFT](#).
- `poseInAnchorSpace` is an [XrPosef](#) defining the position and orientation of the new space's origin relative to the anchor's natural origin.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_anchor](#) extension **must** be enabled prior to using [XrSpatialAnchorSpaceCreateInfoMSFT](#)
- `type` **must** be `XR_TYPE_SPATIAL_ANCHOR_SPACE_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `anchor` **must** be a valid [XrSpatialAnchorMSFT](#) handle

The [xrDestroySpatialAnchorMSFT](#) function is defined as:

```
// Provided by XR_MSFT_spatial_anchor
XrResult xrDestroySpatialAnchorMSFT(
    XrSpatialAnchorMSFT          anchor);
```

## Parameter Descriptions

- `anchor` is a handle to an [XrSpatialAnchorMSFT](#) previously created by [xrCreateSpatialAnchorMSFT](#).

[XrSpatialAnchorMSFT](#) handles are destroyed using [xrDestroySpatialAnchorMSFT](#). By destroying an anchor, the runtime **can** stop spending resources used to maintain tracking for that anchor's origin.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor` extension **must** be enabled prior to calling `xrDestroySpatialAnchorMSFT`
- `anchor` **must** be a valid `XrSpatialAnchorMSFT` handle

## Thread Safety

- Access to `anchor`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## New Object Types

`XrSpatialAnchorMSFT`

## New Flag Types

## New Enum Constants

`XrObjectType` enumeration is extended with:

- `XR_OBJECT_TYPE_SPATIAL_ANCHOR_MSFT`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_MSFT`
- `XR_TYPE_SPATIAL_ANCHOR_SPACE_CREATE_INFO_MSFT`

`XrResult` enumeration is extended with:

- `XR_ERROR_CREATE_SPATIAL_ANCHOR_FAILED_MSFT`

## New Enums

## New Structures

[XrSpatialAnchorCreateInfoMSFT](#)

[XrSpatialAnchorSpaceCreateInfoMSFT](#)

## New Functions

[xrCreateSpatialAnchorMSFT](#)

[xrCreateSpatialAnchorSpaceMSFT](#)

[xrDestroySpatialAnchorMSFT](#)

## Issues

### Version History

- Revision 1, 2019-07-30 (Alex Turner)
  - Initial extension description
- Revision 2, 2021-06-02 (Rylie Pavlik, Collabora, Ltd.)
  - Note that the parameter to [xrDestroySpatialAnchorMSFT](#) must be externally synchronized

# 12.122. XR\_MSFT\_spatial\_anchor\_persistence

## Name String

[XR\\_MSFT\\_spatial\\_anchor\\_persistence](#)

## Extension Type

Instance extension

## Registered Extension Number

143

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_MSFT\\_spatial\\_anchor](#)

## Last Modified Date

2021-07-15

## IP Status

No known IP claims.

## Contributors

Lachlan Ford, Microsoft

Yin Li, Microsoft

Norman Pohl, Microsoft

Alex Turner, Microsoft

Bryce Hutchings, Microsoft

### 12.122.1. Overview

This extension allows persistence and retrieval of spatial anchors sharing and localization across application sessions on a device. Spatial anchors persisted during an application session on a device will only be able to be retrieved during sessions of that same application on the same device. This extension requires [XR\\_MSFT\\_spatial\\_anchor](#) to also be enabled.

### 12.122.2. Spatial Anchor Store Connection

The [XrSpatialAnchorStoreConnectionMSFT](#) handle represents a connection to the spatial anchor store and is used by the application to perform operations on the spatial anchor store such as:

- Persisting and unpersisting of spatial anchors.
- Enumeration of currently persisted anchors.
- Clearing the spatial anchor store of all anchors.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XR_DEFINE_HANDLE(XrSpatialAnchorStoreConnectionMSFT)
```

The application **can** use the [xrCreateSpatialAnchorStoreConnectionMSFT](#) function to create an handle to the spatial anchor store. The application **can** use this handle to interact with the spatial anchor store in order to persist anchors across application sessions.

The [xrCreateSpatialAnchorStoreConnectionMSFT](#) function **may** be a slow operation and therefore **should** be invoked from a non-timing critical thread.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrCreateSpatialAnchorStoreConnectionMSFT(
    XrSession session,
    XrSpatialAnchorStoreConnectionMSFT* spatialAnchorStore);
```

## Parameter Descriptions

- `session` is the `XrSession` the anchor was created with.
- `spatialAnchorStore` is a pointer to the `XrSpatialAnchorStoreConnectionMSFT` handle.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to calling `xrCreateSpatialAnchorStoreConnectionMSFT`
- `session` **must** be a valid `XrSession` handle
- `spatialAnchorStore` **must** be a pointer to an `XrSpatialAnchorStoreConnectionMSFT` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`

The application **can** use the `xrDestroySpatialAnchorStoreConnectionMSFT` function to destroy an anchor store connection.



```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrDestroySpatialAnchorStoreConnectionMSFT(
    XrSpatialAnchorStoreConnectionMSFT    spatialAnchorStore);
```

## Parameter Descriptions

- `spatialAnchorStore` is the [XrSpatialAnchorStoreConnectionMSFT](#) to be destroyed.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_anchor\\_persistence](#) extension **must** be enabled prior to calling [xrDestroySpatialAnchorStoreConnectionMSFT](#)
- `spatialAnchorStore` **must** be a valid [XrSpatialAnchorStoreConnectionMSFT](#) handle

## Thread Safety

- Access to `spatialAnchorStore`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_OUT_OF_MEMORY`

### 12.122.3. Persist Spatial Anchor

The application **can** use the [xrPersistSpatialAnchorMSFT](#) function to persist a spatial anchor in the spatial anchor store for this application. The given [XrSpatialAnchorPersistenceInfoMSFT::spatialAnchorPersistenceName](#) will be the string to retrieve the spatial anchor from the Spatial Anchor store or subsequently remove the record of this spatial anchor from the store. This name will uniquely identify the spatial anchor for the current application. If there is already a spatial anchor of the same name persisted in the spatial anchor store, the existing spatial anchor will be replaced and [xrPersistSpatialAnchorMSFT](#) **must** return `XR_SUCCESS`.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrPersistSpatialAnchorMSFT(
    XrSpatialAnchorStoreConnectionMSFT    spatialAnchorStore,
    const XrSpatialAnchorPersistenceInfoMSFT* spatialAnchorPersistenceInfo);
```

## Parameter Descriptions

- `spatialAnchorStore` is the `XrSpatialAnchorStoreConnectionMSFT` with which to persist the `XrSpatialAnchorPersistenceInfoMSFT::spatialAnchor`.
- `spatialAnchorPersistenceInfo` is a pointer to `XrSpatialAnchorPersistenceInfoMSFT` structure to specify the anchor and its name to persist.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to calling `xrPersistSpatialAnchorMSFT`
- `spatialAnchorStore` **must** be a valid `XrSpatialAnchorStoreConnectionMSFT` handle
- `spatialAnchorPersistenceInfo` **must** be a pointer to a valid `XrSpatialAnchorPersistenceInfoMSFT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT`

The [XrSpatialAnchorPersistenceNameMSFT](#) structure is the name associated with the [XrSpatialAnchorMSFT](#) in the spatial anchor store. It is used to perform persist and unpersist on an `name` in the spatial anchor store.

The [XrSpatialAnchorPersistenceNameMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_spatial_anchor_persistence
typedef struct XrSpatialAnchorPersistenceNameMSFT {
    char    name[XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_MSFT];
} XrSpatialAnchorPersistenceNameMSFT;
```

### Member Descriptions

- `name` is a null terminated character array of size `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_MSFT`.

If an [XrSpatialAnchorPersistenceNameMSFT](#) with an empty `name` value is passed to any function as a parameter, that function **must** return `XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT`.

### Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_anchor\\_persistence](#) extension **must** be enabled prior to using [XrSpatialAnchorPersistenceNameMSFT](#)
- `name` **must** be a null-terminated UTF-8 string whose length is less than or equal to `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_MSFT`

The [XrSpatialAnchorPersistenceInfoMSFT](#) structure is defined as:

```
// Provided by XR_MSFT_spatial_anchor_persistence
typedef struct XrSpatialAnchorPersistenceInfoMSFT {
    XrStructureType          type;
    const void*              next;
    XrSpatialAnchorPersistenceNameMSFT  spatialAnchorPersistenceName;
    XrSpatialAnchorMSFT      spatialAnchor;
} XrSpatialAnchorPersistenceInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `spatialAnchorPersistenceName` is an [XrSpatialAnchorPersistenceNameMSFT](#) containing the name associated with the [XrSpatialAnchorMSFT](#) in the spatial anchor store.
- `spatialAnchor` is the [XrSpatialAnchorMSFT](#) that the application wishes to perform persistence operations on.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_anchor\\_persistence](#) extension **must** be enabled prior to using [XrSpatialAnchorPersistenceInfoMSFT](#)
- `type` **must** be `XR_TYPE_SPATIAL_ANCHOR_PERSISTENCE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `spatialAnchorPersistenceName` **must** be a valid [XrSpatialAnchorPersistenceNameMSFT](#) structure
- `spatialAnchor` **must** be a valid [XrSpatialAnchorMSFT](#) handle

The application **can** use the [xrEnumeratePersistedSpatialAnchorNamesMSFT](#) function to enumerate the names of all spatial anchors currently persisted in the spatial anchor store for this application. This function follows the [two-call idiom](#) for filling the `spatialAnchorNames`.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrEnumeratePersistedSpatialAnchorNamesMSFT(
    XrSpatialAnchorStoreConnectionMSFT    spatialAnchorStore,
    uint32_t                               spatialAnchorNameCapacityInput,
    uint32_t*                              spatialAnchorNameCountOutput,
    XrSpatialAnchorPersistenceNameMSFT*   spatialAnchorNames);
```

## Parameter Descriptions

- `spatialAnchorStore` is the `XrSpatialAnchorStoreConnectionMSFT` anchor store to perform the enumeration operation on.
- `spatialAnchorNameCapacityInput` is the capacity of the `spatialAnchorNames` array, or `0` to indicate a request to retrieve the required capacity.
- `spatialAnchorNameCountOutput` is filled in by the runtime with the count of anchor names written or the required capacity in the case that `spatialAnchorNameCapacityInput` is insufficient.
- `spatialAnchorNames` is a pointer to an array of `XrSpatialAnchorPersistenceNameMSFT` structures, but **can** be `NULL` if `propertyCapacityInput` is `0`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `spatialAnchorNames` size.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to calling `xrEnumeratePersistedSpatialAnchorNamesMSFT`
- `spatialAnchorStore` **must** be a valid `XrSpatialAnchorStoreConnectionMSFT` handle
- `spatialAnchorNameCountOutput` **must** be a pointer to a `uint32_t` value
- If `spatialAnchorNameCapacityInput` is not `0`, `spatialAnchorNames` **must** be a pointer to an array of `spatialAnchorNameCapacityInput` `XrSpatialAnchorPersistenceNameMSFT` structures

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`

The application **can** use the `xrCreateSpatialAnchorFromPersistedNameMSFT` function to create a `XrSpatialAnchorMSFT` from the spatial anchor store. If the `XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT::spatialAnchorPersistenceName` provided does not correspond to a currently stored anchor (i.e. the list of spatial anchor names returned from `xrEnumeratePersistedSpatialAnchorNamesMSFT`), the function **must** return `XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT`.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrCreateSpatialAnchorFromPersistedNameMSFT(
    XrSession session,
    const XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT* spatialAnchorCreateInfo,
    XrSpatialAnchorMSFT* spatialAnchor);
```

## Parameter Descriptions

- `session` is a handle to an `XrSession` previously created with `xrCreateSession`.
- `spatialAnchorCreateInfo` is a pointer to the `XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT`.
- `spatialAnchor` is a pointer to an `XrSpatialAnchorMSFT` handle that will be set by the runtime on successful load.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to calling `xrCreateSpatialAnchorFromPersistedNameMSFT`
- `session` **must** be a valid `XrSession` handle
- `spatialAnchorCreateInfo` **must** be a pointer to a valid `XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT` structure
- `spatialAnchor` **must** be a pointer to an `XrSpatialAnchorMSFT` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT`

The `XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT` structure is defined as:

```
// Provided by XR_MSFT_spatial_anchor_persistence
typedef struct XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT {
    XrStructureType                type;
    const void*                    next;
    XrSpatialAnchorStoreConnectionMSFT spatialAnchorStore;
    XrSpatialAnchorPersistenceNameMSFT spatialAnchorPersistenceName;
} XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR.
- `spatialAnchorStore` is the `XrSpatialAnchorStoreConnectionMSFT` from which the spatial anchor will be loaded from.
- `spatialAnchorPersistenceName` is the `XrSpatialAnchorPersistenceNameMSFT` associated with the `XrSpatialAnchorMSFT` in the spatial anchor store. This name is used to create an `XrSpatialAnchorMSFT` handle from a spatial anchor persisted in the spatial anchor store.

The `spatialAnchorPersistenceName` is a character array of maximum size `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_MSFT`, which **must** include a null terminator and **must** not be empty (i.e. the first element is the null terminator). If an empty `spatialAnchorPersistenceName` value is passed to any function as a parameter, that function **must** return `XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to using `XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT`
- `type` **must** be `XR_TYPE_SPATIAL_ANCHOR_FROM_PERSISTED_ANCHOR_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `spatialAnchorStore` **must** be a valid `XrSpatialAnchorStoreConnectionMSFT` handle
- `spatialAnchorPersistenceName` **must** be a valid `XrSpatialAnchorPersistenceNameMSFT` structure

The application **can** use the `xrUnpersistSpatialAnchorMSFT` function to remove the record of the anchor in the spatial anchor store. This operation will not affect any `XrSpatialAnchorMSFT` handles previously created. If the `spatialAnchorPersistenceName` provided does not correspond to a currently stored anchor, the function **must** return `XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT`.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrUnpersistSpatialAnchorMSFT(
    XrSpatialAnchorStoreConnectionMSFT    spatialAnchorStore,
    const XrSpatialAnchorPersistenceNameMSFT* spatialAnchorPersistenceName);
```



## Parameter Descriptions

- `spatialAnchorStore` is an `XrSpatialAnchorStoreConnectionMSFT` anchor store to perform the unpersist operation on.
- `spatialAnchorPersistenceName` is a pointer to the `XrSpatialAnchorPersistenceNameMSFT`.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to calling `xrUnpersistSpatialAnchorMSFT`
- `spatialAnchorStore` **must** be a valid `XrSpatialAnchorStoreConnectionMSFT` handle
- `spatialAnchorPersistenceName` **must** be a pointer to a valid `XrSpatialAnchorPersistenceNameMSFT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT`

The application **can** use the `xrClearSpatialAnchorStoreMSFT` function to remove all spatial anchors from the spatial anchor store for this application. The function only removes the record of the spatial anchors in the store but does not affect any `XrSpatialAnchorMSFT` handles previously loaded in the current session.

```
// Provided by XR_MSFT_spatial_anchor_persistence
XrResult xrClearSpatialAnchorStoreMSFT(
    XrSpatialAnchorStoreConnectionMSFT    spatialAnchorStore);
```

## Parameter Descriptions

- `spatialAnchorStore` is `XrSpatialAnchorStoreConnectionMSFT` to perform the clear operation on.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_anchor_persistence` extension **must** be enabled prior to calling `xrClearSpatialAnchorStoreMSFT`
- `spatialAnchorStore` **must** be a valid `XrSpatialAnchorStoreConnectionMSFT` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

## New Object Types

- `XrSpatialAnchorStoreConnectionMSFT`

## New Flag Types

## New Enum Constants

- `XR_TYPE_SPATIAL_ANCHOR_PERSISTENCE_INFO_MSFT`

- `XR_TYPE_SPATIAL_ANCHOR_FROM_PERSISTED_ANCHOR_CREATE_INFO_MSFT`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_NOT_FOUND_MSFT`
- `XR_ERROR_SPATIAL_ANCHOR_NAME_INVALID_MSFT`
- `XR_MAX_SPATIAL_ANCHOR_NAME_SIZE_MSFT`

### New Enums

### New Structures

- `XrSpatialAnchorPersistenceNameMSFT`
- `XrSpatialAnchorPersistenceInfoMSFT`
- `XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT`

### New Functions

- `xrCreateSpatialAnchorStoreConnectionMSFT`
- `xrDestroySpatialAnchorStoreConnectionMSFT`
- `xrPersistSpatialAnchorMSFT`
- `xrEnumeratePersistedSpatialAnchorNamesMSFT`
- `xrCreateSpatialAnchorFromPersistedNameMSFT`
- `xrUnpersistSpatialAnchorMSFT`
- `xrClearSpatialAnchorStoreMSFT`

### Version History

- Revision 1, 2021-02-19 (Lachlan Ford)
  - Initial extension proposal
- Revision 2, 2021-07-15 (Yin Li)
  - Extension proposal to OpenXR working group

## 12.123. XR\_MSFT\_spatial\_graph\_bridge

### Name String

`XR_MSFT_spatial_graph_bridge`

### Extension Type

Instance extension

### Registered Extension Number

50

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Darryl Gough, Microsoft

Yin Li, Microsoft

Alex Turner, Microsoft

David Fields, Microsoft

## Overview

This extension enables applications to interop between [XrSpace](#) handles and other Windows Mixed Reality device platform libraries or APIs. These libraries represent a spatially tracked point, also known as a "spatial graph node", with a GUID value. This extension enables applications to create [XrSpace](#) handles from spatial graph nodes. Applications can also try to get a spatial graph node from an [XrSpace](#) handle.

### 12.123.1. Create [XrSpace](#) from Spatial Graph Node

The `xrCreateSpatialGraphNodeSpaceMSFT` function creates an [XrSpace](#) handle for a given spatial graph node type and ID.

```
// Provided by XR_MSFT_spatial_graph_bridge
XrResult xrCreateSpatialGraphNodeSpaceMSFT(
    XrSession session,
    const XrSpatialGraphNodeSpaceCreateInfoMSFT* createInfo,
    XrSpace* space);
```

### Parameter Descriptions

- `session` is the [XrSession](#) which will use the created space.
- `createInfo` is an [XrSpatialGraphNodeSpaceCreateInfoMSFT](#) specifying the space to be created.
- `space` is the returned [XrSpace](#) handle for the given spatial node ID.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_graph_bridge` extension **must** be enabled prior to calling `xrCreateSpatialGraphNodeSpaceMSFT`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrSpatialGraphNodeSpaceCreateInfoMSFT` structure
- `space` **must** be a pointer to an `XrSpace` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`

The `XrSpatialGraphNodeSpaceCreateInfoMSFT` structure is used with `xrCreateSpatialGraphNodeSpaceMSFT` to create an `XrSpace` handle for a given spatial node type and node ID.

```
// Provided by XR_MSFT_spatial_graph_bridge
typedef struct XrSpatialGraphNodeSpaceCreateInfoMSFT {
    XrStructureType          type;
    const void*              next;
    XrSpatialGraphNodeTypeMSFT  nodeType;
    uint8_t                  nodeId[XR_GUID_SIZE_MSFT];
    XrPosef                  pose;
} XrSpatialGraphNodeSpaceCreateInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `nodeType` is an [XrSpatialGraphNodeTypeMSFT](#) specifying the spatial node type.
- `nodeId` is a global unique identifier (a.k.a. GUID or 16 byte array), representing the spatial node that is being tracked.
- `pose` is an [XrPosef](#) defining the position and orientation of the new space's origin within the natural reference frame of the spatial graph node.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_graph\\_bridge](#) extension **must** be enabled prior to using [XrSpatialGraphNodeSpaceCreateInfoMSFT](#)
- `type` **must** be `XR_TYPE_SPATIAL_GRAPH_NODE_SPACE_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `nodeType` **must** be a valid [XrSpatialGraphNodeTypeMSFT](#) value

The enum [XrSpatialGraphNodeTypeMSFT](#) describes the types of spatial graph nodes.

```
// Provided by XR_MSFT_spatial_graph_bridge
typedef enum XrSpatialGraphNodeTypeMSFT {
    XR_SPATIAL_GRAPH_NODE_TYPE_STATIC_MSFT = 1,
    XR_SPATIAL_GRAPH_NODE_TYPE_DYNAMIC_MSFT = 2,
    XR_SPATIAL_GRAPH_NODE_TYPE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSpatialGraphNodeTypeMSFT;
```

There are two types of spatial graph nodes: static and dynamic.

Static spatial nodes track the pose of a fixed location in the world relative to reference spaces. The tracking of static nodes **may** slowly adjust the pose over time for better accuracy but the pose is relatively stable in the short term, such as between rendering frames. For example, a QR code tracking library can use a static node to represent the location of the tracked QR code. Static spatial nodes are represented by `XR_SPATIAL_GRAPH_NODE_TYPE_STATIC_MSFT`.

Dynamic spatial nodes track the pose of a physical object that moves continuously relative to reference spaces. The pose of dynamic spatial nodes **can** be very different within the duration of a rendering frame. It is important for the application to use the correct timestamp to query the space location using `XrLocateSpace`. For example, a color camera mounted in front of a HMD is also tracked by the HMD so a web camera library can use a dynamic node to represent the camera location. Dynamic spatial nodes are represented by `XR_SPATIAL_GRAPH_NODE_TYPE_DYNAMIC_MSFT`.

### 12.123.2. Create Spatial Graph Node Binding from `XrSpace`

The `XrSpatialGraphNodeBindingMSFT` handle represents a binding to a spatial graph node. This handle allows an application to get a spatial graph node GUID from an `XrSpace` to use in other Windows Mixed Reality device platform libraries or APIs.

The runtime **must** remember the spatial graph node and track it for the lifetime of the `XrSpatialGraphNodeBindingMSFT` handle. When the `XrSpatialGraphNodeBindingMSFT` handle is destroyed then the runtime's tracking system **may** forget about the spatial graphic node and stop tracking it.

```
XR_DEFINE_HANDLE(XrSpatialGraphNodeBindingMSFT)
```

The `xrTryCreateSpatialGraphStaticNodeBindingMSFT` function tries to create a binding to the best spatial graph static node relative to the given location and returns an `XrSpatialGraphNodeBindingMSFT` handle.

```
// Provided by XR_MSFT_spatial_graph_bridge
XrResult xrTryCreateSpatialGraphStaticNodeBindingMSFT(
    XrSession session,
    const XrSpatialGraphStaticNodeBindingCreateInfoMSFT* createInfo,
    XrSpatialGraphNodeBindingMSFT* nodeBinding);
```

## Parameter Descriptions

- `session` is the specified [XrSession](#).
- `createInfo` is the [XrSpatialGraphStaticNodeBindingCreateInfoMSFT](#) input structure.
- `nodeBinding` is the [XrSpatialGraphNodeBindingMSFT](#) output structure.

The runtime **may** return `XR_SUCCESS` and set `nodeBinding` to `XR_NULL_HANDLE` if it is unable to create a spatial graph static node binding. This may happen when the given [XrSpace](#) cannot be properly tracked at the moment. The application can retry creating the [XrSpatialGraphNodeBindingMSFT](#) handle again after a reasonable period of time when tracking is regained.

The [xrTryCreateSpatialGraphStaticNodeBindingMSFT](#) function **may** be a slow operation and therefore **should** be invoked from a non-timing critical thread.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_graph_bridge` extension **must** be enabled prior to calling [xrTryCreateSpatialGraphStaticNodeBindingMSFT](#)
- `session` **must** be a valid [XrSession](#) handle
- `createInfo` **must** be a pointer to a valid [XrSpatialGraphStaticNodeBindingCreateInfoMSFT](#) structure
- `nodeBinding` **must** be a pointer to an [XrSpatialGraphNodeBindingMSFT](#) handle



## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_OUT\_OF\_MEMORY
- XR\_ERROR\_LIMIT\_REACHED
- XR\_ERROR\_TIME\_INVALID
- XR\_ERROR\_POSE\_INVALID

`XrSpatialGraphStaticNodeBindingCreateInfoMSFT` is an input structure for `xrTryCreateSpatialGraphStaticNodeBindingMSFT`.

```
// Provided by XR_MSFT_spatial_graph_bridge
typedef struct XrSpatialGraphStaticNodeBindingCreateInfoMSFT {
    XrStructureType    type;
    const void*        next;
    XrSpace             space;
    XrPosef            poseInSpace;
    XrTime             time;
} XrSpatialGraphStaticNodeBindingCreateInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `space` is a handle to the [XrSpace](#) in which `poseInSpace` is specified.
- `poseInSpace` is the [XrPosef](#) within `space` at `time`.
- `time` is the [XrTime](#) at which `poseInSpace` will be evaluated within `space`.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_graph\\_bridge](#) extension **must** be enabled prior to using [XrSpatialGraphStaticNodeBindingCreateInfoMSFT](#)
- `type` **must** be `XR_TYPE_SPATIAL_GRAPH_STATIC_NODE_BINDING_CREATE_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `space` **must** be a valid [XrSpace](#) handle

The [xrDestroySpatialGraphNodeBindingMSFT](#) function releases the `nodeBinding` and the underlying resources.

```
// Provided by XR_MSFT_spatial_graph_bridge
XrResult xrDestroySpatialGraphNodeBindingMSFT(
    XrSpatialGraphNodeBindingMSFT          nodeBinding);
```

## Parameter Descriptions

- `nodeBinding` is an [XrSpatialGraphNodeBindingMSFT](#) previously created by [xrTryCreateSpatialGraphStaticNodeBindingMSFT](#).

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_graph\\_bridge](#) extension **must** be enabled prior to calling [xrDestroySpatialGraphNodeBindingMSFT](#)
- `nodeBinding` **must** be a valid [XrSpatialGraphNodeBindingMSFT](#) handle

## Thread Safety

- Access to `nodeBinding`, and any child handles, **must** be externally synchronized

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_HANDLE_INVALID`

## Get spatial graph node binding properties

The `xrGetSpatialGraphNodeBindingPropertiesMSFT` function retrieves the spatial graph node GUID and the pose in the node space from an `XrSpatialGraphNodeBindingMSFT` handle.

```
// Provided by XR_MSFT_spatial_graph_bridge
XrResult xrGetSpatialGraphNodeBindingPropertiesMSFT(
    XrSpatialGraphNodeBindingMSFT          nodeBinding,
    const XrSpatialGraphNodeBindingPropertiesGetInfoMSFT* getInfo,
    XrSpatialGraphNodeBindingPropertiesMSFT* properties);
```

## Parameter Descriptions

- `nodeBinding` is an `XrSpatialGraphNodeBindingMSFT` previously created by `xrTryCreateSpatialGraphStaticNodeBindingMSFT`.
- `getInfo` is a pointer to an `XrSpatialGraphNodeBindingPropertiesGetInfoMSFT` input structure.
- `properties` is a pointer to an `XrSpatialGraphNodeBindingPropertiesMSFT` output structure.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_graph_bridge` extension **must** be enabled prior to calling `xrGetSpatialGraphNodeBindingPropertiesMSFT`
- `nodeBinding` **must** be a valid `XrSpatialGraphNodeBindingMSFT` handle
- If `getInfo` is not `NULL`, `getInfo` **must** be a pointer to a valid `XrSpatialGraphNodeBindingPropertiesGetInfoMSFT` structure
- `properties` **must** be a pointer to an `XrSpatialGraphNodeBindingPropertiesMSFT` structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`

`XrSpatialGraphNodeBindingPropertiesGetInfoMSFT` is an input structure for `xrGetSpatialGraphNodeBindingPropertiesMSFT`.

```
// Provided by XR_MSFT_spatial_graph_bridge
typedef struct XrSpatialGraphNodeBindingPropertiesGetInfoMSFT {
    XrStructureType    type;
    const void*        next;
} XrSpatialGraphNodeBindingPropertiesGetInfoMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.

## Valid Usage (Implicit)

- The [XR\\_MSFT\\_spatial\\_graph\\_bridge](#) extension **must** be enabled prior to using [XrSpatialGraphNodeBindingPropertiesGetInfoMSFT](#)
- `type` **must** be `XR_TYPE_SPATIAL_GRAPH_NODE_BINDING_PROPERTIES_GET_INFO_MSFT`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

[XrSpatialGraphNodeBindingPropertiesMSFT](#) is an output structure for [xrGetSpatialGraphNodeBindingPropertiesMSFT](#).

```
// Provided by XR_MSFT_spatial_graph_bridge
typedef struct XrSpatialGraphNodeBindingPropertiesMSFT {
    XrStructureType    type;
    void*              next;
    uint8_t            nodeId[XR_GUID_SIZE_MSFT];
    XrPosef            poseInNodeSpace;
} XrSpatialGraphNodeBindingPropertiesMSFT;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `nodeId` is a global unique identifier (a.k.a. GUID or 16 byte array), representing the spatial graph node.
- `poseInNodeSpace` is an [XrPosef](#) defining the pose in the underlying node's space.

## Valid Usage (Implicit)

- The `XR_MSFT_spatial_graph_bridge` extension **must** be enabled prior to using `XrSpatialGraphNodeBindingPropertiesMSFT`
- `type` **must** be `XR_TYPE_SPATIAL_GRAPH_NODE_BINDING_PROPERTIES_MSFT`
- `next` **must** be `NULL` or a valid pointer to the `next` structure in a structure chain

### New Object Types

- `XrSpatialGraphNodeBindingMSFT`

### New Flag Types

### New Enum Constants

`XrObjectType` enumeration is extended with:

- `XR_OBJECT_TYPE_SPATIAL_GRAPH_NODE_BINDING_MSFT`

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SPATIAL_GRAPH_NODE_SPACE_CREATE_INFO_MSFT`
- `XR_TYPE_SPATIAL_GRAPH_STATIC_NODE_BINDING_CREATE_INFO_MSFT`
- `XR_TYPE_SPATIAL_GRAPH_NODE_BINDING_PROPERTIES_GET_INFO_MSFT`
- `XR_TYPE_SPATIAL_GRAPH_NODE_BINDING_PROPERTIES_MSFT`

### New Enums

- `XrSpatialGraphNodeTypeMSFT`

### New Structures

- `XrSpatialGraphNodeSpaceCreateInfoMSFT`
- `XrSpatialGraphStaticNodeBindingCreateInfoMSFT`
- `XrSpatialGraphNodeBindingPropertiesGetInfoMSFT`
- `XrSpatialGraphNodeBindingPropertiesMSFT`

### New Functions

- `xrTryCreateSpatialGraphStaticNodeBindingMSFT`
- `xrDestroySpatialGraphNodeBindingMSFT`
- `xrGetSpatialGraphNodeBindingPropertiesMSFT`

## Issues

### Version History

- Revision 1, 2019-10-31 (Yin LI)
  - Initial extension description
- Revision 2, 2022-01-13 (Darryl Gough)
  - Added Spatial Graph Node Binding handle.

## 12.124. XR\_MSFT\_unbounded\_reference\_space

### Name String

`XR_MSFT_unbounded_reference_space`

### Extension Type

Instance extension

### Registered Extension Number

39

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Overview

This extension allows an application to create an `UNBOUNDED_MSFT` reference space. This reference space enables the viewer to move freely through a complex environment, often many meters from where they started, while always optimizing for coordinate system stability near the viewer. This is done by allowing the origin of the reference space to drift as necessary to keep the viewer's coordinates relative to the space's origin stable.

To create an `UNBOUNDED_MSFT` reference space, the application **can** pass `XR_REFERENCE_SPACE_TYPE_UNBOUNDED_MSFT` to `xrCreateReferenceSpace`.

The `UNBOUNDED_MSFT` reference space establishes a world-locked origin, gravity-aligned to exclude pitch and roll, with +Y up, +X to the right, and -Z forward. This space begins with an arbitrary initial position and orientation, which the runtime **may** define to be either the initial position at app launch or some other initial zero position. Unlike a `STAGE` reference space, the runtime **may** place the origin of an `UNBOUNDED_MSFT` reference space at any height, rather than fixing it at the floor. This is because the viewer may move through various rooms and levels of their environment, each of which has a different floor height. Runtimes **should** not automatically adjust the position of the origin when the

viewer moves to a room with a different floor height.

`UNBOUNDED_MSFT` space is useful when an app needs to render **world-scale** content that spans beyond the bounds of a single `STAGE`, for example, an entire floor or multiple floors of a building.

An `UNBOUNDED_MSFT` space maintains stability near the viewer by slightly adjusting its origin over time. The runtime **must** not queue the `XrEventDataReferenceSpaceChangePending` event in response to these minor adjustments.

When views, controllers or other spaces experience tracking loss relative to the `UNBOUNDED_MSFT` space, runtimes **should** continue to provide inferred or last-known `position` and `orientation` values. These inferred poses can, for example, be based on neck model updates, inertial dead reckoning, or a last-known position, so long as it is still reasonable for the application to use that pose. While a runtime is providing position data, it **must** continue to set `XR_SPACE_LOCATION_POSITION_VALID_BIT` and `XR_VIEW_STATE_POSITION_VALID_BIT` but it **can** clear `XR_SPACE_LOCATION_POSITION_TRACKED_BIT` and `XR_VIEW_STATE_POSITION_TRACKED_BIT` to indicate that the position is inferred or last-known in this way.

When tracking is recovered, runtimes **should** snap the pose of other spaces back into position relative to the `UNBOUNDED_MSFT` space's original origin. However, if tracking recovers into a new tracking volume in which the original origin can no longer be located (e.g. the viewer moved through a dark hallway and regained tracking in a new room), the runtime **may** recenter the origin arbitrarily, for example moving the origin to coincide with the viewer. If such recentering occurs, the runtime **must** queue the `XrEventDataReferenceSpaceChangePending` event with `poseValid` set to false.

If the viewer moves far enough away from the origin of an `UNBOUNDED_MSFT` reference space that floating point error would introduce noticeable error when locating the viewer within that space, the runtime **may** recenter the space's origin to a new location closer to the viewer. If such recentering occurs, the runtime **must** queue the `XrEventDataReferenceSpaceChangePending` event with `poseValid` set to true.

Runtimes **must** support the `UNBOUNDED_MSFT` reference space when this extension is enabled.

## New Object Types

## New Flag Types

## New Enum Constants

`XrReferenceSpaceType` enumeration is extended with:

- `XR_REFERENCE_SPACE_TYPE_UNBOUNDED_MSFT`

## New Enums

## New Structures

## New Functions

## Issues



## Version History

- Revision 1, 2019-07-30 (Alex Turner)
  - Initial extension description

# 12.125. XR\_OCULUS\_audio\_device\_guid

## Name String

`XR_OCULUS_audio_device_guid`

## Extension Type

Instance extension

## Registered Extension Number

160

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Overview

This extension enables the querying of audio device information associated with an OpenXR instance.

On Windows, there may be multiple audio devices available on the system. This extensions allows applications to query the runtime for the appropriate audio devices for the active HMD.

## New Object Types

## New Flag Types

## New Enum Constants

- `XR_MAX_AUDIO_DEVICE_STR_SIZE_OCULUS`

## New Enums

## New Structures

## New Functions

```
// Provided by XR_OCULUS_audio_device_guid
XrResult xrGetAudioOutputDeviceGuidOculus(
    XrInstance          instance,
    wchar_t             buffer
[XR_MAX_AUDIO_DEVICE_STR_SIZE_OCULUS]);
```

## Parameter Descriptions

- **instance** is the [XrInstance](#) to query the audio device state in.
- **buffer** is a fixed size buffer which will contain the audio device GUID. The format of this data matches the [IMMDevice::GetId](#) API.

## Valid Usage (Implicit)

- The [XR\\_OCULUS\\_audio\\_device\\_guid](#) extension **must** be enabled prior to calling [xrGetAudioOutputDeviceGuidOculus](#)
- **instance** **must** be a valid [XrInstance](#) handle
- **buffer** **must** be a wide character array of length [XR\\_MAX\\_AUDIO\\_DEVICE\\_STR\\_SIZE\\_OCULUS](#)

## Return Codes

### Success

- [XR\\_SUCCESS](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

```
// Provided by XR_OCULUS_audio_device_guid
XrResult xrGetAudioInputDeviceGuidOculus(
    XrInstance          instance,
    wchar_t             buffer
[XR_MAX_AUDIO_DEVICE_STR_SIZE_OCULUS]);
```

## Parameter Descriptions

- **instance** is the [XrInstance](#) to query the audio device state in.
- **buffer** is a fixed size buffer which will contain the audio device GUID. The format of this data matches the [IMMDevice::GetId](#) API.

## Valid Usage (Implicit)

- The [XR\\_OCULUS\\_audio\\_device\\_guid](#) extension **must** be enabled prior to calling [xrGetAudioInputDeviceGuidOculus](#)
- **instance** **must** be a valid [XrInstance](#) handle
- **buffer** **must** be a wide character array of length [XR\\_MAX\\_AUDIO\\_DEVICE\\_STR\\_SIZE\\_OCULUS](#)

## Return Codes

### Success

- [XR\\_SUCCESS](#)

### Failure

- [XR\\_ERROR\\_FUNCTION\\_UNSUPPORTED](#)
- [XR\\_ERROR\\_VALIDATION\\_FAILURE](#)
- [XR\\_ERROR\\_RUNTIME\\_FAILURE](#)
- [XR\\_ERROR\\_HANDLE\\_INVALID](#)
- [XR\\_ERROR\\_INSTANCE\\_LOST](#)
- [XR\\_ERROR\\_FEATURE\\_UNSUPPORTED](#)

## Issues

## Version History

- Revision 1, 2021-05-13 (John Kearney)

- Initial extension description

## 12.126. XR\_OCULUS\_external\_camera

### Name String

`XR_OCULUS_external_camera`

### Extension Type

Instance extension

### Registered Extension Number

227

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Overview

This extension enables the querying of external camera information for a session. This extension is intended to enable mixed reality capture support for applications.

This extension does not provide a mechanism for supplying external camera information to the runtime. If external camera information is not supplied to the runtime before using this extension, no camera information will be returned.

This API supports returning camera intrinsics and extrinsics:

- Camera intrinsics are the attributes of the camera: resolution, field of view, etc.
- Camera extrinsics are everything external to the camera: relative pose, attached to, etc.
- We do not expect the camera intrinsics to change frequently. We expect the camera extrinsics to change frequently.

### New Object Types

### New Flag Types

```
typedef XrFlags64 XrExternalCameraStatusFlagsOCULUS;
```

```

// Flag bits for XrExternalCameraStatusFlagsOCULUS
static const XrExternalCameraStatusFlagsOCULUS
XR_EXTERNAL_CAMERA_STATUS_CONNECTED_BIT_OCULUS = 0x00000001;
static const XrExternalCameraStatusFlagsOCULUS
XR_EXTERNAL_CAMERA_STATUS_CALIBRATING_BIT_OCULUS = 0x00000002;
static const XrExternalCameraStatusFlagsOCULUS
XR_EXTERNAL_CAMERA_STATUS_CALIBRATION_FAILED_BIT_OCULUS = 0x00000004;
static const XrExternalCameraStatusFlagsOCULUS
XR_EXTERNAL_CAMERA_STATUS_CALIBRATED_BIT_OCULUS = 0x00000008;
static const XrExternalCameraStatusFlagsOCULUS
XR_EXTERNAL_CAMERA_STATUS_CAPTURING_BIT_OCULUS = 0x00000010;

```

## Flag Descriptions

- `XR_EXTERNAL_CAMERA_STATUS_CONNECTED_BIT_OCULUS` — External camera is connected
- `XR_EXTERNAL_CAMERA_STATUS_CALIBRATING_BIT_OCULUS` — External camera is undergoing calibration
- `XR_EXTERNAL_CAMERA_STATUS_CALIBRATION_FAILED_BIT_OCULUS` — External camera has tried and failed calibration
- `XR_EXTERNAL_CAMERA_STATUS_CALIBRATED_BIT_OCULUS` — External camera has tried and passed calibration
- `XR_EXTERNAL_CAMERA_STATUS_CAPTURING_BIT_OCULUS` — External camera is capturing

## New Enum Constants

`XR_MAX_EXTERNAL_CAMERA_NAME_SIZE_OCULUS` defines the length of the field `XrExternalCameraOCULUS::name`.

```
#define XR_MAX_EXTERNAL_CAMERA_NAME_SIZE_OCULUS 32
```

`XrStructureType` enumeration is extended with:

- `XR_TYPE_EXTERNAL_CAMERA_OCULUS`

## New Enums

```
// Provided by XR_OCULUS_external_camera
typedef enum XrExternalCameraAttachedToDeviceOCULUS {
    XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_NONE_OCULUS = 0,
    XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_HMD_OCULUS = 1,
    XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_LTOUCH_OCULUS = 2,
    XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_RTOUCH_OCULUS = 3,
    XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_MAX_ENUM_OCULUS = 0x7FFFFFFF
} XrExternalCameraAttachedToDeviceOCULUS;
```

| Enum   | Description   |
|--|---|
| <code>XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_NONE_OCULUS</code>   | External camera is at a fixed point in LOCAL space      |
| <code>XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_HMD_OCULUS</code>    | External camera is attached to the HMD                  |
| <code>XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_LTOUCH_OCULUS</code> | External camera is attached to a left Touch controller  |
| <code>XR_EXTERNAL_CAMERA_ATTACHED_TO_DEVICE_RTOUCH_OCULUS</code> | External camera is attached to a right Touch controller |

## New Structures

The `XrExternalCameraIntrinsicsOCULUS` structure is defined as:

```
// Provided by XR_OCULUS_external_camera
typedef struct XrExternalCameraIntrinsicsOCULUS {
    XrTime        lastChangeTime;
    XrFovf        fov;
    float         virtualNearPlaneDistance;
    float         virtualFarPlaneDistance;
    XrExtent2Di   imageSensorPixelResolution;
} XrExternalCameraIntrinsicsOCULUS;
```

## Member Descriptions

- `lastChangeTime` is the `XrTime` when this camera's intrinsics last changed.
- `fov` is the `XrFovf` 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` specifying the camera's resolution (in pixels).

## Valid Usage (Implicit)

- The `XR_OCULUS_external_camera` extension **must** be enabled prior to using `XrExternalCameraIntrinsicsOCULUS`

The `XrExternalCameraExtrinsicsOCULUS` structure is defined as:

```
// Provided by XR_OCULUS_external_camera
typedef struct XrExternalCameraExtrinsicsOCULUS {
    XrTime                lastChangeTime;
    XrExternalCameraStatusFlagsOCULUS    cameraStatusFlags;
    XrExternalCameraAttachedToDeviceOCULUS    attachedToDevice;
    XrPosef                relativePose;
} XrExternalCameraExtrinsicsOCULUS;
```

## Member Descriptions

- `lastChangeTime` is the `XrTime` when this camera's extrinsics last changed.
- `cameraStatusFlags` is the `XrExternalCameraStatusFlagsOCULUS` for this camera's status.
- `attachedToDevice` is the `XrExternalCameraAttachedToDeviceOCULUS` for the device this camera is attached to
- `relativePose` is the `XrPosef` for offset of the camera from the device that the camera is attached to

## Valid Usage (Implicit)

- The `XR_OCULUS_external_camera` extension **must** be enabled prior to using `XrExternalCameraExtrinsicsOCULUS`
- `cameraStatusFlags` **must** be `0` or a valid combination of `XrExternalCameraStatusFlagBitsOCULUS` values
- `attachedToDevice` **must** be a valid `XrExternalCameraAttachedToDeviceOCULUS` value

The `XrExternalCameraOCULUS` structure is defined as:

```
// Provided by XR_OCULUS_external_camera
typedef struct XrExternalCameraOCULUS {
    XrStructureType          type;
    const void*              next;
    char                     name[XR_MAX_EXTERNAL_CAMERA_NAME_SIZE_OCULUS];
    XrExternalCameraIntrinsicsOCULUS  intrinsics;
    XrExternalCameraExtrinsicsOCULUS  extrinsics;
} XrExternalCameraOCULUS;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `name` is a null-terminated UTF-8 string containing a camera identifier: VID (vendor ID), PID (product ID), and serial number
- `intrinsics` is the `XrExternalCameraIntrinsicsOCULUS` for the camera
- `extrinsics` is the `XrExternalCameraExtrinsicsOCULUS` for the camera

## Valid Usage (Implicit)

- The `XR_OCULUS_external_camera` extension **must** be enabled prior to using `XrExternalCameraOCULUS`
- `type` **must** be `XR_TYPE_EXTERNAL_CAMERA_OCULUS`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)



## New Functions

The `xrEnumerateExternalCamerasOCULUS` function enumerates all the external cameras that are supported by the runtime, it is defined as:

```
// Provided by XR_OCULUS_external_camera
XrResult xrEnumerateExternalCamerasOCULUS(
    XrSession session,
    uint32_t cameraCapacityInput,
    uint32_t* cameraCountOutput,
    XrExternalCameraOCULUS* cameras);
```

### Parameter Descriptions

- `session` is the `XrSession` to query the external cameras in
- `cameraCapacityInput` is the capacity of the `cameras` array, or 0 to indicate a request to retrieve the required capacity.
- `cameraCountOutput` is filled in by the runtime with the count of `cameras` written or the required capacity in the case that `cameraCapacityInput` is insufficient.
- `cameras` is an array of `XrExternalCameraOCULUS` filled in by the runtime which contains all the available external cameras, but **can** be `NULL` if `cameraCapacityInput` is 0.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `cameras` size.

### Valid Usage (Implicit)

- The `XR_OCULUS_external_camera` extension **must** be enabled prior to calling `xrEnumerateExternalCamerasOCULUS`
- `session` **must** be a valid `XrSession` handle
- `cameraCountOutput` **must** be a pointer to a `uint32_t` value
- If `cameraCapacityInput` is not 0, `cameras` **must** be a pointer to an array of `cameraCapacityInput` `XrExternalCameraOCULUS` structures

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_SIZE_INSUFFICIENT`

## Issues

## Version History

- Revision 1, 2022-08-31 (John Kearney)
  - Initial extension description

## 12.127. XR\_OPPO\_controller\_interaction

### Name String

`XR_OPPO_controller_interaction`

### Extension Type

Instance extension

### Registered Extension Number

454

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

## Contributors

Haomiao Jiang, OPPO  
Buyi Xu, OPPO  
Yebao Cai, OPPO

## Overview

This extension defines a new interaction profile for the OPPO Controller, including but not limited to OPPO MR Glasses Controller.

### OPPO Controller interaction profile

Interaction profile path:

- */interaction\_profiles/oppo/mr\_controller\_oppo*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the OPPO Controller.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
  - *.../input/hearttrate\_oppo/value*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
  - *.../input/home/click* (**may** not be available for application use)
- *.../input/squeeze/value*
- *.../input/trigger/touch*
- *.../input/trigger/value*

- `.../input/grip/pose`
- `.../input/aim/pose`
- `.../input/thumbstick/click`
- `.../input/thumbstick/touch`
- `.../input/thumbstick`
- `.../input/thumbstick/x`
- `.../input/thumbstick/y`
- `.../output/haptic`

## New Identifiers

- **heartrate\_oppo**: OPPO MR Controller adds an optional heart rate sensor to monitor the heart beat rate of the user.

## Input Path Descriptions

- `/input/hearttrate_oppo/value` : Allow developers to access the heart beat per minute (BPM) of the user. The data would only be available with user's active consent.

### Note



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`

### Note



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`

### Note



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



#### Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

### Version History

- Revision 1, Haomiao Jiang
  - Initial extension description

## 12.128. XR\_QCOM\_tracking\_optimization\_settings

### Name String

`XR_QCOM_tracking_optimization_settings`

### Extension Type

Instance extension

### Registered Extension Number

307

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2022-06-02

### Contributors

Daniel Guttenberg, Qualcomm  
Martin Renschler, Qualcomm  
Karthik Nagarajan, Qualcomm

### Overview

This extension defines an API for the application to give optimization hints to the runtime for tracker domains.

For example, an application might be interested in tracking targets that are at a far distance from the camera which **may** increase tracking latency, while another application might be interested in

minimizing power consumption at the cost of tracking accuracy. Targets are domains which are defined in [XrTrackingOptimizationSettingsDomainQCOM](#).

This allows the application to tailor the tracking algorithms to specific use-cases and scene-scales in order to provide the best experience possible.

Summary: provide domain hints to the run-time about which parameters to optimize tracking for.

### 12.128.1. Setting Tracking Optimization Hints

The tracking optimization hints are expressed as a hint [XrTrackingOptimizationSettingsHintQCOM](#).

```
// Provided by XR_QCOM_tracking_optimization_settings
typedef enum XrTrackingOptimizationSettingsDomainQCOM {
    XR_TRACKING_OPTIMIZATION_SETTINGS_DOMAIN_ALL_QCOM = 1,
    XR_TRACKING_OPTIMIZATION_SETTINGS_DOMAIN_MAX_ENUM_QCOM = 0x7FFFFFFF
} XrTrackingOptimizationSettingsDomainQCOM;
```

#### Enumerant Descriptions

- `XR_TRACKING_OPTIMIZATION_SETTINGS_DOMAIN_ALL_QCOM` — Setting applies to all QCOM tracking extensions.

```
// Provided by XR_QCOM_tracking_optimization_settings
typedef enum XrTrackingOptimizationSettingsHintQCOM {
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_NONE_QCOM = 0,
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_LONG_RANGE_PRIORIZATION_QCOM = 1,
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_CLOSE_RANGE_PRIORIZATION_QCOM = 2,
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_LOW_POWER_PRIORIZATION_QCOM = 3,
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_HIGH_POWER_PRIORIZATION_QCOM = 4,
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_MAX_ENUM_QCOM = 0x7FFFFFFF
} XrTrackingOptimizationSettingsHintQCOM;
```

## Enumerant Descriptions

- `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_NONE_QCOM` — Used by the application to indicate that it does not have a preference to optimize for. The run-time is understood to choose a balanced approach.
- `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_LONG_RANGE_PRIORIZATION_QCOM` — Used by the application to indicate that it prefers tracking to be optimized for long range, possibly at the expense of competing interests.
- `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_CLOSE_RANGE_PRIORIZATION_QCOM` — Used by the application to indicate that it prefers tracking to be optimized for close range, possibly at the expense of competing interests.
- `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_LOW_POWER_PRIORIZATION_QCOM` — Used by the application to indicate that it prefers tracking to be optimized for low power consumption, possibly at the expense of competing interests.
- `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_HIGH_POWER_PRIORIZATION_QCOM` — Used by the application to indicate that it prefers tracking to be optimized for increased tracking performance, possibly at the cost of increased power consumption.

The `xrSetTrackingOptimizationSettingsHintQCOM` function is defined as:

```
// Provided by XR_QCOM_tracking_optimization_settings
XrResult xrSetTrackingOptimizationSettingsHintQCOM(
    XrSession session,
    XrTrackingOptimizationSettingsDomainQCOM domain,
    XrTrackingOptimizationSettingsHintQCOM hint);
```

## Parameter Descriptions

- `session` is a valid `XrSession` handle.
- `domain` is the tracking domain for which the hint is applied
- `hint` is the hint to be applied

The XR runtime behaves as if `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_NONE_QCOM` was submitted if the application does not provide a hint.

The XR runtime **must** return `XR_ERROR_VALIDATION_FAILURE` if the application sets a domain or hint not part of `XrTrackingOptimizationSettingsDomainQCOM` or `XrTrackingOptimizationSettingsHintQCOM`.

A hint is typically set before a domain handle is created. If hints are set more than once from one or concurrent sessions, the runtime **may** accommodate the first hint it received and return `XR_ERROR_HINT_ALREADY_SET_QCOM` for any subsequent calls made.

If the application destroys the active domain handle associated with the hint, the runtime **may** behave as if `XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_NONE_QCOM` was set. In this scenario, the runtime **should** accommodate new valid hints that **may** be set for the same domain.

### Valid Usage (Implicit)

- The `XR_QCOM_tracking_optimization_settings` extension **must** be enabled prior to calling `xrSetTrackingOptimizationSettingsHintQCOM`
- `session` **must** be a valid `XrSession` handle
- `domain` **must** be a valid `XrTrackingOptimizationSettingsDomainQCOM` value
- `hint` **must** be a valid `XrTrackingOptimizationSettingsHintQCOM` value

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_HINT_ALREADY_SET_QCOM`

## 12.128.2. Example of setting a tracking optimization hint



```
XrInstance instance; // previously initialized
XrSession session;  // previously initialized

// Get function pointer for xrSetTrackingOptimizationSettingsHintQCOM
PFN_xrSetTrackingOptimizationSettingsHintQCOM pfnSetTrackingOptimizationSettingsHintQCOM;
CHK_XR(xrGetInstanceProcAddr(instance, "xrSetTrackingOptimizationSettingsHintQCOM",
    (PFN_xrVoidFunction*)&pfnSetTrackingOptimizationSettingsHintQCOM));

pfnSetTrackingOptimizationSettingsHintQCOM(session,
    XR_TRACKING_OPTIMIZATION_SETTINGS_DOMAIN_ALL_QCOM,
    XR_TRACKING_OPTIMIZATION_SETTINGS_HINT_LONG_RANGE_PRIORIZATION_QCOM);

// perform tracking while prioritizing long range tracking
```

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

- [XrTrackingOptimizationSettingsHintQCOM](#)
- [XrTrackingOptimizationSettingsDomainQCOM](#)

## New Structures

## New Functions

- [xrSetTrackingOptimizationSettingsHintQCOM](#)

## Issues

## Version History

- Revision 1, 2022-06-02
  - Initial extension description

# 12.129. XR\_ULTRALEAP\_hand\_tracking\_forearm

## Name String

`XR_ULTRALEAP_hand_tracking_forearm`

## Extension Type

Instance extension

## Registered Extension Number

150

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_EXT\\_hand\\_tracking](#)

## Last Modified Date

2022-04-19

## IP Status

No known IP claims.

## Contributors

Robert Blenkinsopp, Ultraleap  
Adam Harwood, Ultraleap

## Overview

This extension augments the [XR\\_EXT\\_hand\\_tracking](#) extension to enable applications to request the default set of 26 hand joints, with the addition of a joint representing the user's elbow.

The application **must** also enable the [XR\\_EXT\\_hand\\_tracking](#) extension in order to use this extension.

## New joint set

This extension extends the [XrHandJointSetEXT](#) enumeration with a new member [XR\\_HAND\\_JOINT\\_SET\\_HAND\\_WITH\\_FOREARM\\_ULTRALEAP](#). This joint set is the same as the [XR\\_HAND\\_JOINT\\_SET\\_DEFAULT\\_EXT](#), plus a joint representing the user's elbow, [XR\\_HAND\\_FOREARM\\_JOINT\\_ELLOW\\_ULTRALEAP](#).

```
// Provided by XR_ULTRALEAP_hand_tracking_forearm
typedef enum XrHandForearmJointULTRALEAP {
    XR_HAND_FOREARM_JOINT_PALM_ULTRALEAP = 0,
    XR_HAND_FOREARM_JOINT_WRIST_ULTRALEAP = 1,
    XR_HAND_FOREARM_JOINT_THUMB_METACARPAL_ULTRALEAP = 2,
    XR_HAND_FOREARM_JOINT_THUMB_PROXIMAL_ULTRALEAP = 3,
    XR_HAND_FOREARM_JOINT_THUMB_DISTAL_ULTRALEAP = 4,
    XR_HAND_FOREARM_JOINT_THUMB_TIP_ULTRALEAP = 5,
    XR_HAND_FOREARM_JOINT_INDEX_METACARPAL_ULTRALEAP = 6,
    XR_HAND_FOREARM_JOINT_INDEX_PROXIMAL_ULTRALEAP = 7,
    XR_HAND_FOREARM_JOINT_INDEX_INTERMEDIATE_ULTRALEAP = 8,
    XR_HAND_FOREARM_JOINT_INDEX_DISTAL_ULTRALEAP = 9,
    XR_HAND_FOREARM_JOINT_INDEX_TIP_ULTRALEAP = 10,
    XR_HAND_FOREARM_JOINT_MIDDLE_METACARPAL_ULTRALEAP = 11,
    XR_HAND_FOREARM_JOINT_MIDDLE_PROXIMAL_ULTRALEAP = 12,
    XR_HAND_FOREARM_JOINT_MIDDLE_INTERMEDIATE_ULTRALEAP = 13,
    XR_HAND_FOREARM_JOINT_MIDDLE_DISTAL_ULTRALEAP = 14,
    XR_HAND_FOREARM_JOINT_MIDDLE_TIP_ULTRALEAP = 15,
    XR_HAND_FOREARM_JOINT_RING_METACARPAL_ULTRALEAP = 16,
    XR_HAND_FOREARM_JOINT_RING_PROXIMAL_ULTRALEAP = 17,
    XR_HAND_FOREARM_JOINT_RING_INTERMEDIATE_ULTRALEAP = 18,
    XR_HAND_FOREARM_JOINT_RING_DISTAL_ULTRALEAP = 19,
    XR_HAND_FOREARM_JOINT_RING_TIP_ULTRALEAP = 20,
    XR_HAND_FOREARM_JOINT_LITTLE_METACARPAL_ULTRALEAP = 21,
    XR_HAND_FOREARM_JOINT_LITTLE_PROXIMAL_ULTRALEAP = 22,
    XR_HAND_FOREARM_JOINT_LITTLE_INTERMEDIATE_ULTRALEAP = 23,
    XR_HAND_FOREARM_JOINT_LITTLE_DISTAL_ULTRALEAP = 24,
    XR_HAND_FOREARM_JOINT_LITTLE_TIP_ULTRALEAP = 25,
    XR_HAND_FOREARM_JOINT_ELBOW_ULTRALEAP = 26,
    XR_HAND_FOREARM_JOINT_MAX_ENUM_ULTRALEAP = 0x7FFFFFFF
} XrHandForearmJointULTRALEAP;
```



*Note*

The first [XR\\_HAND\\_JOINT\\_COUNT\\_EXT](#) members of [XrHandForearmJointULTRALEAP](#) are identical to the members of [XrHandJointEXT](#) and **can** be used interchangeably.

The [XR\\_HAND\\_FOREARM\\_JOINT\\_ELBOW\\_ULTRALEAP](#) joint represents the center of an elbow and is orientated with the backwards (+Z) direction parallel to the forearm and points away from the hand.

The up (+Y) direction is pointing out of the dorsal side of the forearm. The X direction is perpendicular to Y and Z and follows the right hand rule.

```
// Provided by XR_ULTRALEAP_hand_tracking_forearm
#define XR_HAND_FOREARM_JOINT_COUNT_ULTRALEAP 27
```

[XR\\_HAND\\_FOREARM\\_JOINT\\_COUNT\\_ULTRALEAP](#) defines the number of hand joint enumerants defined in [XrHandForearmJointULTRALEAP](#).

### New Object Types

### New Flag Types

### New Enum Constants

- [XR\\_HAND\\_FOREARM\\_JOINT\\_COUNT\\_ULTRALEAP](#)

[XrHandJointSetEXT](#) enumeration is extended with:

- [XR\\_HAND\\_JOINT\\_SET\\_HAND\\_WITH\\_FOREARM\\_ULTRALEAP](#)

### New Enums

- [XrHandForearmJointULTRALEAP](#)

### New Structures

### New Functions

### Issues

### Version History

- Revision 1, 2022-04-19 (Robert Blenkinsopp)
  - Initial version

## 12.130. XR\_VALVE\_analog\_threshold

### Name String

[XR\\_VALVE\\_analog\\_threshold](#)

### Extension Type

Instance extension

### Registered Extension Number

80

### Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-06-09

## IP Status

No known IP claims.

## Contributors

Joe Ludwig, Valve

Rune Berg, Valve

Andres Rodriguez, Valve

## Overview

This extension allows the application to control the threshold and haptic feedback applied to an analog to digital conversion. See [XrInteractionProfileAnalogThresholdVALVE](#) for more information.

Applications **should** also enable the [XR\\_KHR\\_binding\\_modification](#) extension to be able to define multiple thresholds.

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

The [XrInteractionProfileAnalogThresholdVALVE](#) structure is an input struct that defines thresholds and haptic feedback behavior for action bindings and **should** be added to the [XrBindingModificationsKHR::bindingModifications](#) array of the [XrBindingModificationsKHR](#) structure (See [XR\\_KHR\\_binding\\_modification](#) extension).

```
// Provided by XR_VALVE_analog_threshold
typedef struct XrInteractionProfileAnalogThresholdVALVE {
    XrStructureType          type;
    const void*              next;
    XrAction                  action;
    XrPath                    binding;
    float                     onThreshold;
    float                     offThreshold;
    const XrHapticBaseHeader* onHaptic;
    const XrHapticBaseHeader* offHaptic;
} XrInteractionProfileAnalogThresholdVALVE;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `action` is the handle of an action in the suggested binding list.
- `binding` is the input path used for the specified action in the suggested binding list.
- `onThreshold` is the value between 0.0 and 1.0 at which the runtime **must** consider the binding to be true. The binding must remain true until the input analog value falls below `offThreshold`.
- `offThreshold` is the value between 0.0 and 1.0 at which the runtime **must** consider the binding to be false if it was previous true.
- `onHaptic` is the haptic output that the runtime **must** trigger when the binding changes from false to true. If this field is `NULL`, the runtime **must** not trigger any haptic output on the threshold. This field **can** point to any supported sub-type of [XrHapticBaseHeader](#).
- `offHaptic` is the haptic output that the runtime **must** trigger when the binding changes from true to false. If this field is `NULL`, the runtime **must** not trigger any haptic output on the threshold. This field **can** point to any supported sub-type of [XrHapticBaseHeader](#).

Applications can also chain a single [XrInteractionProfileAnalogThresholdVALVE](#) structure on the next chain of any [xrSuggestInteractionProfileBindings](#) call. Runtimes **must** support this kind of chaining. This method of specifying analog thresholds is deprecated however, and should not be used by any new applications.

If a threshold struct is present for a given conversion, the runtime **must** use those thresholds instead of applying its own whenever it is using the binding suggested by the application.

`onThreshold` and `offThreshold` permit allow the application to specify that it wants hysteresis to be applied to the threshold operation. If `onThreshold` is smaller than `offThreshold`, the runtime **must**

return `XR_ERROR_VALIDATION_FAILURE`.

`onHaptic` and `offHaptic` allow the application to specify that it wants automatic haptic feedback to be generated when the boolean output of the threshold operation changes from false to true or vice versa. If these fields are not NULL, the runtime **must** trigger a haptic output with the specified characteristics. If the device has multiple haptic outputs, the runtime **should** use the haptic output that is most appropriate for the specified input path.

If a suggested binding with `action` and `binding` is not in the binding list for this interaction profile, the runtime **must** return `XR_ERROR_PATH_UNSUPPORTED`.

### Valid Usage (Implicit)

- The `XR_VALVE_analog_threshold` extension **must** be enabled prior to using `XrInteractionProfileAnalogThresholdVALVE`
- `type` **must** be `XR_TYPE_INTERACTION_PROFILE_ANALOG_THRESHOLD_VALVE`
- `next` **must** be NULL or a valid pointer to the [next structure in a structure chain](#)
- `action` **must** be a valid `XrAction` handle
- If `onHaptic` is not NULL, `onHaptic` **must** be a pointer to a valid `XrHapticBaseHeader`-based structure. See also: [XrHapticAmplitudeEnvelopeVibrationFB](#), [XrHapticPcmVibrationFB](#), [XrHapticVibration](#)
- If `offHaptic` is not NULL, `offHaptic` **must** be a pointer to a valid `XrHapticBaseHeader`-based structure. See also: [XrHapticAmplitudeEnvelopeVibrationFB](#), [XrHapticPcmVibrationFB](#), [XrHapticVibration](#)

## New Functions

## Issues

## Version History

- Revision 1, 2020-06-29 (Joe Ludwig)
  - Initial version.
- Revision 2, 2021-07-28 (Rune Berg)
  - Deprecate chaining of struct in [XrInteractionProfileSuggestedBinding](#), applications should use [XrBindingModificationsKHR](#) defined in the `XR_KHR_binding_modification` extension instead.

## 12.131. XR\_VARJO\_composition\_layer\_depth\_test

### Name String

`XR_VARJO_composition_layer_depth_test`

## Extension Type

Instance extension

## Registered Extension Number

123

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_KHR\\_composition\\_layer\\_depth](#)

## Last Modified Date

2021-07-15

## IP Status

No known IP claims.

## Contributors

Sergiy Dubovik, Varjo Technologies

Antti Hirvonen, Varjo Technologies

Rémi Arnaud, Varjo Technologies

## Overview

This extension enables depth-based layer composition inside the compositor.

Core OpenXR specifies that layer compositing must happen in the layer submission order (as described in [Compositing](#)). However, an application may want to composite the final image against the other layers based on depth information for proper occlusion. Layers can now provide depth information that will be used to calculate occlusion between those layers, as well as with the environment depth estimator ([XR\\_VARJO\\_environment\\_depth\\_estimation](#)) when enabled.

This extension defines a new type, [XrCompositionLayerDepthTestVARJO](#), which can be chained to [XrCompositionLayerProjection](#) in order to activate this functionality. An application must also specify a range where depth testing will happen, potentially covering only a subset of the full depth range.

## Composition

Layer composition rules change when this extension is enabled.

If the application does not chain [XrCompositionLayerDepthTestVARJO](#), "painter's algorithm" such as described in [Compositing](#) must be used for layer composition.



Overall, composition should be performed in the following way:

1. Layers must be composited in the submission order. The compositor must track the depth value nearest to the virtual camera. Initial value for the nearest depth should be infinity.
2. If the currently processed layer does not contain depth, compositor should composite the layer against the previous layers with "painter's algorithm" and move to the next layer.
3. If the layer depth or the active nearest depth fall inside the depth test range of the layer, the compositor must perform depth test against the layer and active depth. If the layer depth is less or equal than the active depth, layer is composited normally with the previous layers and active depth is updated to match the layer depth. Otherwise the layer pixel is discarded, and compositor should move to composite the next layer.

## Example

Mixed reality applications may want to show hands on top of the rendered VR content. For this purpose the application should enable environment depth estimation (see [XR\\_VARJO\\_environment\\_depth\\_estimation](#) extension) and depth testing with range 0m to 1m.

The following code illustrates how to enable depth testing:

```
XrCompositionLayerProjection layer; // previously populated

XrCompositionLayerDepthTestVARJO depthTest{XR_TYPE_COMPOSITION_LAYER_DEPTH_TEST_VARJO,
layer.next};
depthTest.depthTestRangeNearZ = 0.0f; // in meters
depthTest.depthTestRangeFarZ = 1.0f; // in meters
layer.next = &depthTest;
```

## New Structures

Applications **can** enable depth testing by adding [XrCompositionLayerDepthTestVARJO](#) to the **next** chain for all [XrCompositionLayerProjectionView](#) structures in the given layer in addition to [XrCompositionLayerDepthInfoKHR](#). Missing [XrCompositionLayerDepthInfoKHR](#) automatically disables the depth testing functionality.

The [XrCompositionLayerDepthTestVARJO](#) structure is defined as:

```
// Provided by XR_VARJO_composition_layer_depth_test
typedef struct XrCompositionLayerDepthTestVARJO {
    XrStructureType    type;
    const void*        next;
    float               depthTestRangeNearZ;
    float               depthTestRangeFarZ;
} XrCompositionLayerDepthTestVARJO;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `depthTestRangeNearZ` is a non-negative distance in meters that specifies the lower bound of the range where depth testing should be performed. Must be less than `depthTestRangeFarZ`. Value of zero means that there is no lower bound.
- `depthTestRangeFarZ` is a positive distance in meters that specifies the upper bound of the range where depth testing should be performed. Must be greater than `depthTestRangeNearZ`. Value of floating point positive infinity means that there is no upper bound.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_composition\\_layer\\_depth\\_test](#) extension **must** be enabled prior to using [XrCompositionLayerDepthTestVARJO](#)
- `type` **must** be `XR_TYPE_COMPOSITION_LAYER_DEPTH_TEST_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_COMPOSITION_LAYER_DEPTH_TEST_VARJO`

## Version History

- Revision 1, 2021-02-16 (Sergiy Dubovik)
  - Initial extension description
- Revision 2, 2021-07-15 (Rylie Pavlik, Collabora, Ltd., and Sergiy Dubovik)
  - Update sample code so it is buildable

# 12.132. XR\_VARJO\_environment\_depth\_estimation

## Name String

`XR_VARJO_environment_depth_estimation`

## Extension Type

Instance extension

## Registered Extension Number

124

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-02-17

## IP Status

No known IP claims.

## Contributors

Sergiy Dubovik, Varjo Technologies  
Antti Hirvonen, Varjo Technologies  
Rémi Arnaud, Varjo Technologies

## Overview

This extension provides a mechanism for enabling depth estimation of the environment in the runtime-supplied compositor. This is an extension to `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` mode to not only use the color but also depth for composition of the final image.

Mixed reality applications might want to mix real and virtual content based on the depth information for proper occlusion. XR hardware and runtime may offer various ways to estimate the depth of the environment inside the compositor. When this estimation is enabled, the compositor can generate properly occluded final image when layers are submitted with depth information (both [XR\\_KHR\\_composition\\_layer\\_depth](#) and [XR\\_VARJO\\_composition\\_layer\\_depth\\_test](#)).

This extension defines a new function, `xrSetEnvironmentDepthEstimationVARJO`, which **can** be used to toggle environment depth estimation in the compositor. Toggling depth estimation is an asynchronous operation and the feature **may** not be activated immediately. Function can be called immediately after the session is created. Composition of the environment layer follows the rules as described in [XR\\_VARJO\\_composition\\_layer\\_depth\\_test](#).

## New Structures

The `xrSetEnvironmentDepthEstimationVARJO` function is defined as:

```
// Provided by XR_VARJO_environment_depth_estimation
XrResult xrSetEnvironmentDepthEstimationVARJO(
    XrSession session,
    XrBool32 enabled);
```

### Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `enabled` is a boolean that specifies whether depth estimation functionality should be activated. Compositor will disable depth estimation functionality if environment blend mode is not `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` and will enable the functionality when environment blend mode is set to `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND`.

### Valid Usage (Implicit)

- The `XR_VARJO_environment_depth_estimation` extension **must** be enabled prior to calling `xrSetEnvironmentDepthEstimationVARJO`
- `session` **must** be a valid `XrSession` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## New Functions

## Version History

- Revision 1, 2021-02-16 (Sergiy Dubovik)
  - Initial extension description

## 12.133. XR\_VARJO\_foveated\_rendering

### Name String

`XR_VARJO_foveated_rendering`

### Extension Type

Instance extension

### Registered Extension Number

122

### Revision

3

### Extension and Version Dependencies

[OpenXR 1.0](#)

and

[XR\\_VARJO\\_quad\\_views](#)

## Last Modified Date

2021-04-13

## IP Status

No known IP claims.

## Contributors

Sergiy Dubovik, Varjo Technologies

Rémi Arnaud, Varjo Technologies

Antti Hirvonen, Varjo Technologies

### 12.133.1. Overview

Varjo headsets provide extremely high pixel density displays in the center area of the display, blended with a high density display covering the rest of the field of view. If the application has to provide a single image per eye, that would cover the entire field of view, at the highest density it would be extremely resource intensive, and in fact impossible for the most powerful desktop GPUs to render in real time. So instead Varjo introduced the [XR\\_VARJO\\_quad\\_views](#) extension enabling the application to provide two separate images for the two screen areas, resulting in a significant reduction in processing, for pixels that could not even be seen.

This extension goes a step further by enabling the application to only generate the density that can be seen by the user, which is another big reduction compared to the density that can be displayed, using dedicated eye tracking.

This extension requires [XR\\_VARJO\\_quad\\_views](#) extension to be enabled.

An application using this extension to enable foveated rendering will take the following steps to prepare:

1. Enable [XR\\_VARJO\\_quad\\_views](#) and [XR\\_VARJO\\_foveated\\_rendering](#) extensions.
2. Query system properties in order to determine if system supports foveated rendering.
3. Query texture sizes for foveated rendering.

In the render loop, for each frame, an application using this extension **should**

1. Check if rendering gaze is available using [xrLocateSpace](#).
2. Enable foveated rendering when [xrLocateViews](#) is called.

### 12.133.2. Inspect system capability

An application **can** inspect whether the system is capable of foveated rendering by chaining an [XrSystemFoveatedRenderingPropertiesVARJO](#) structure to the [XrSystemProperties](#) structure when calling [xrGetSystemProperties](#).

```
// Provided by XR_VARJO_foveated_rendering
typedef struct XrSystemFoveatedRenderingPropertiesVARJO {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsFoveatedRendering;
} XrSystemFoveatedRenderingPropertiesVARJO;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `supportsFoveatedRendering` is an [XrBool32](#), indicating if current system is capable of performing foveated rendering.

The runtime **should** return `XR_TRUE` for `supportsFoveatedRendering` when rendering gaze is available in the system. An application **should** avoid using foveated rendering functionality when `supportsFoveatedRendering` is `XR_FALSE`.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_foveated\\_rendering](#) extension **must** be enabled prior to using [XrSystemFoveatedRenderingPropertiesVARJO](#)
- `type` **must** be `XR_TYPE_SYSTEM_FOVEATED_RENDERING_PROPERTIES_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### 12.133.3. Determine foveated texture sizes

Foveated textures **may** have different sizes and aspect ratio compared to non-foveated textures. In order to determine recommended foveated texture size, an application **can** chain [XrFoveatedViewConfigurationViewVARJO](#) to [XrViewConfigurationView](#) and set `foveatedRenderingActive` to `XR_TRUE`. Since an application using foveated rendering with this extension has to render four views, [XR\\_VARJO\\_quad\\_views](#) **must** be enabled along with this extension when [XrInstance](#) is created.

First and second views are non foveated views (covering whole field of view of HMD), third (left eye) and fourth (right eye) are foveated e.g. following gaze.

```
// Provided by XR_VARJO_foveated_rendering
typedef struct XrFoveatedViewConfigurationViewVARJO {
    XrStructureType    type;
    void*              next;
    XrBool32           foveatedRenderingActive;
} XrFoveatedViewConfigurationViewVARJO;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `foveatedRenderingActive` is an [XrBool32](#), indicating if the runtime should return foveated view configuration view.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_foveated\\_rendering](#) extension **must** be enabled prior to using [XrFoveatedViewConfigurationViewVARJO](#)
- `type` **must** be `XR_TYPE_FOVEATED_VIEW_CONFIGURATION_VIEW_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

For example:



```

XrInstance instance; // previously populated
XrSystemId systemId; // previously populated
XrViewConfigurationType viewConfigType; // Select
XR_VIEW_CONFIGURATION_TYPE_PRIMARY_QUAD_VARJO

XrSystemFoveatedRenderingPropertiesVARJO foveatedRenderingProperties
{XR_TYPE_SYSTEM_FOVEATED_RENDERING_PROPERTIES_VARJO};
XrSystemProperties systemProperties{XR_TYPE_SYSTEM_PROPERTIES,
&foveatedRenderingProperties};
CHK_XR(xrGetSystemProperties(instance, systemId, &systemProperties));

uint32_t viewCount;
CHK_XR(xrEnumerateViewConfigurationViews(instance, systemId, viewConfigType, 0,
&viewCount, nullptr));
// Non-foveated rendering views dimensions
std::vector<XrViewConfigurationView> configViews(viewCount,
{XR_TYPE_VIEW_CONFIGURATION_VIEW});
CHK_XR(xrEnumerateViewConfigurationViews(instance, systemId, viewConfigType, viewCount,
&viewCount, configViews.data()));

// Foveated rendering views dimensions
std::vector<XrViewConfigurationView> foveatedViews;
if (foveatedRenderingProperties.supportsFoveatedRendering && viewConfigType ==
XR_VIEW_CONFIGURATION_TYPE_PRIMARY_QUAD_VARJO) {
    std::vector<XrFoveatedViewConfigurationViewVARJO> requestFoveatedConfig{4,
{XR_TYPE_FOVEATED_VIEW_CONFIGURATION_VIEW_VARJO, nullptr, XR_TRUE}};
    foveatedViews = std::vector<XrViewConfigurationView>{4,
{XR_TYPE_VIEW_CONFIGURATION_VIEW}};
    for (size_t i = 0; i < 4; i++) {
        foveatedViews[i].next = &requestFoveatedConfig[i];
    }
    CHK_XR(xrEnumerateViewConfigurationViews(instance, systemId, viewConfigType, viewCount,
&viewCount, foveatedViews.data()));
}

```

### Example 3. Note

Applications using this extension are encouraged to create two sets of swapchains or one big enough set of swapchains and two sets of viewports. One set will be used when rendering gaze is not available and other one will be used when foveated rendering and rendering gaze is available. Using foveated textures **may** not provide optimal visual quality when rendering gaze is not available.

## 12.133.4. Rendering gaze status

Extension defines new reference space type - `XR_REFERENCE_SPACE_TYPE_COMBINED_EYE_VARJO` which **should** be used to determine whether rendering gaze is available. After calling `xrLocateSpace`, application **should** inspect `XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT` bit. If it's set, rendering gaze is available otherwise not.

```
XrSession session; // previously populated

// Create needed spaces
XrSpace viewSpace;
XrReferenceSpaceCreateInfo createViewSpaceInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};
createViewSpaceInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_VIEW;
createViewSpaceInfo.poseInReferenceSpace.orientation.w = 1.0f;
CHK_XR(xrCreateReferenceSpace(session, &createViewSpaceInfo, &viewSpace));

XrSpace renderGazeSpace;
XrReferenceSpaceCreateInfo createReferenceSpaceInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};
createReferenceSpaceInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_COMBINED_EYE_VARJO;
createReferenceSpaceInfo.poseInReferenceSpace.orientation.w = 1.0f;
CHK_XR(xrCreateReferenceSpace(session, &createReferenceSpaceInfo, &renderGazeSpace));

// ...
// in frame loop
// ...

XrFrameState frameState; // previously populated by xrWaitFrame

// Query rendering gaze status
XrSpaceLocation renderGazeLocation{XR_TYPE_SPACE_LOCATION};
CHK_XR(xrLocateSpace(renderGazeSpace, viewSpace, frameState.predictedDisplayTime,
&renderGazeLocation));

const bool foveationActive = (renderGazeLocation.locationFlags &
XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT) != 0;

if (foveationActive) {
    // Rendering gaze is available
} else {
    // Rendering gaze is not available
}
```

## 12.133.5. Request foveated field of view

For each frame, the application indicates if the runtime will return foveated or non-foveated field of view. This is done by chaining `XrViewLocateFoveatedRenderingVARJO` to `XrViewLocateInfo`.

```
// Provided by XR_VARJO_foveated_rendering
typedef struct XrViewLocateFoveatedRenderingVARJO {
    XrStructureType    type;
    const void*        next;
    XrBool32           foveatedRenderingActive;
} XrViewLocateFoveatedRenderingVARJO;
```

## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `foveatedRenderingActive` is an `XrBool32`, indicating if runtime should return foveated FoV.

The runtime **must** return foveated field of view when `foveatedRenderingActive` is `XR_TRUE`.

## Valid Usage (Implicit)

- The `XR_VARJO_foveated_rendering` extension **must** be enabled prior to using `XrViewLocateFoveatedRenderingVARJO`
- `type` **must** be `XR_TYPE_VIEW_LOCATE_FOVEATED_RENDERING_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

```

// ...
// in frame loop
// ...

XrSession session; // previously populated
XrSpace appSpace; // previously populated
XrFrameState frameState; // previously populated by xrWaitFrame
XrViewConfigurationType viewConfigType; // previously populated
std::vector<XrView> views; // previously populated/resized to the correct size
bool foveationActive; // previously populated, as in the previous example

XrViewState viewState{XR_TYPE_VIEW_STATE};
uint32_t viewCapacityInput = static_cast<uint32_t>(views.size());
uint32_t viewCountOutput;
XrViewLocateInfo viewLocateInfo{XR_TYPE_VIEW_LOCATE_INFO};
viewLocateInfo.viewConfigurationType = viewConfigType;
viewLocateInfo.displayTime = frameState.predictedDisplayTime;
viewLocateInfo.space = appSpace;
XrViewLocateFoveatedRenderingVARJO viewLocateFoveatedRendering
{XR_TYPE_VIEW_LOCATE_FOVEATED_RENDERING_VARJO};
viewLocateFoveatedRendering.foveatedRenderingActive = foveationActive;
viewLocateInfo.next = &viewLocateFoveatedRendering;

CHK_XR(xrLocateViews(session, &viewLocateInfo, &viewState, viewCapacityInput,
&viewCountOutput, views.data()));

```

## New Structures

- [XrViewLocateFoveatedRenderingVARJO](#)
- [XrFoveatedViewConfigurationViewVARJO](#)
- [XrSystemFoveatedRenderingPropertiesVARJO](#)

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_VIEW\\_LOCATE\\_FOVEATED\\_RENDERING\\_VARJO](#)
- [XR\\_TYPE\\_FOVEATED\\_VIEW\\_CONFIGURATION\\_VIEW\\_VARJO](#)
- [XR\\_TYPE\\_SYSTEM\\_FOVEATED\\_RENDERING\\_PROPERTIES\\_VARJO](#)

[XrReferenceSpaceType](#) enumeration is extended with:

- [XR\\_REFERENCE\\_SPACE\\_TYPE\\_COMBINED\\_EYE\\_VARJO](#)

## Version History

- Revision 1, 2020-12-16 (Sergiy Dubovik)
  - Initial extension description
- Revision 2, 2021-04-13 (Rylie Pavlik, Collabora, Ltd., and Sergiy Dubovik)
  - Update sample code so it is buildable
- Revision 3, 2022-02-21 (Denny Rönngren)
  - Update sample code with a missing struct field initialization

## 12.134. XR\_VARJO\_marker\_tracking

### Name String

`XR_VARJO_marker_tracking`

### Extension Type

Instance extension

### Registered Extension Number

125

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2021-09-30

### IP Status

No known IP claims.

### Contributors

Roman Golovanov, Varjo Technologies  
Rémi Arnaud, Varjo Technologies  
Sergiy Dubovik, Varjo Technologies

### 12.134.1. Overview

Varjo Markers are physical markers tracked by the video cameras of the HMD. Different types of markers **can** be used for different purposes. As an example, Varjo Markers **can** be used as cheap replacements for electronic trackers. The cost per printed tracker is significantly lower and the markers require no power to function.

This extension provides the tracking interface to a set of marker types and sizes. Markers **can** be

printed out from the PDF documents and instructions freely available at <https://developer.varjo.com/docs/get-started/varjo-markers#printing-varjo-markers>. Note that the printed marker **must** have the exact physical size for its ID.

Object markers are used to track static or dynamic objects in the user environment. You **may** use object markers in both XR and VR applications. Each marker has a unique ID, and you **must** not use the same physical marker more than once in any given environment. For added precision, an application **may** use multiple markers to track a single object. For example, you could track a monitor by placing a marker in each corner.

There is a set of marker IDs recognized by runtime and if the application uses ID which is not in the set then runtime **must** return `XR_ERROR_MARKER_ID_INVALID_VARJO`.

## New Object Types

## New Flag Types

## New Enums

## New Functions

The `xrSetMarkerTrackingVARJO` function is defined as:

```
// Provided by XR_VARJO_marker_tracking
XrResult xrSetMarkerTrackingVARJO(
    XrSession          session,
    XrBool32           enabled);
```

### Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `enabled` is the flag to enable or disable marker tracking.

The `xrSetMarkerTrackingVARJO` function enables or disables marker tracking functionality. As soon as feature is become disabled all trackable markers become inactive and corresponding events will be generated. An application **may** call any of the functions in this extension regardless if the marker tracking functionality is enabled or disabled.

## Valid Usage (Implicit)

- The `XR_VARJO_marker_tracking` extension **must** be enabled prior to calling `xrSetMarkerTrackingVARJO`
- `session` **must** be a valid `XrSession` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrSetMarkerTrackingTimeoutVARJO` function is defined as:

```
// Provided by XR_VARJO_marker_tracking
XrResult xrSetMarkerTrackingTimeoutVARJO(
    XrSession          session,
    uint64_t           markerId,
    XrDuration         timeout);
```

## Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `markerId` is the unique identifier of the marker for which the timeout will be updated.
- `timeout` is the desired lifetime duration for a specified marker.

The `xrSetMarkerTrackingTimeoutVARJO` function sets a desired lifetime duration for a specified

marker. The default value is `XR_NO_DURATION`. Negative value will be clamped to `XR_NO_DURATION`. It defines the time period during which the runtime **must** keep returning poses of previously tracked markers. The tracking may be lost if the marker went outside of the trackable field of view. In this case the runtime still will try to predict marker's pose for the timeout period. The runtime **must** return `XR_ERROR_MARKER_ID_INVALID_VARJO` if the supplied `markerId` is invalid.

### Valid Usage (Implicit)

- The `XR_VARJO_marker_tracking` extension **must** be enabled prior to calling `xrSetMarkerTrackingTimeoutVARJO`
- `session` **must** be a valid `XrSession` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_MARKER_ID_INVALID_VARJO`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrSetMarkerTrackingPredictionVARJO` function is defined as:

```
// Provided by XR_VARJO_marker_tracking
XrResult xrSetMarkerTrackingPredictionVARJO(
    XrSession          session,
    uint64_t           markerId,
    XrBool32           enable);
```



## Parameter Descriptions

- `session` is an [XrSession](#) handle previously created with [xrCreateSession](#).
- `markerId` is the unique identifier of the marker which should be tracked with prediction.
- `enable` is whether to enable the prediction feature.

The [xrSetMarkerTrackingPredictionVARJO](#) function enables or disables the prediction feature for a specified marker. By default, markers are created with disabled prediction. This works well for markers that are supposed to be stationary. The prediction **can** be used to improve tracking of movable markers. The runtime **must** return `XR_ERROR_MARKER_ID_INVALID_VARJO` if the supplied `markerId` is invalid.

## Valid Usage (Implicit)

- The `XR_VARJO_marker_tracking` extension **must** be enabled prior to calling [xrSetMarkerTrackingPredictionVARJO](#)
- `session` **must** be a valid [XrSession](#) handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_MARKER_ID_INVALID_VARJO`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The [xrGetMarkerSizeVARJO](#) function is defined as:

```
// Provided by XR_VARJO_marker_tracking
XrResult xrGetMarkerSizeVARJO(
    XrSession          session,
    uint64_t           markerId,
    XrExtent2Df*       size);
```

## Parameter Descriptions

- `session` is an [XrSession](#) handle previously created with [xrCreateSession](#).
- `markerId` is the unique identifier of the marker for which size is requested.
- `size` is pointer to the size to populate by the runtime with the physical size of plane marker in meters.

The [xrGetMarkerSizeVARJO](#) function retrieves the height and width of an active marker. The runtime **must** return `XR_ERROR_MARKER_NOT_TRACKED_VARJO` if marker tracking functionality is disabled or the marker with given `markerId` is inactive. The runtime **must** return `XR_ERROR_MARKER_ID_INVALID_VARJO` if the supplied `markerId` is invalid.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_marker\\_tracking](#) extension **must** be enabled prior to calling [xrGetMarkerSizeVARJO](#)
- `session` **must** be a valid [XrSession](#) handle
- `size` **must** be a pointer to an [XrExtent2Df](#) structure

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_MARKER_NOT_TRACKED_VARJO`
- `XR_ERROR_MARKER_ID_INVALID_VARJO`
- `XR_ERROR_FEATURE_UNSUPPORTED`

The `xrCreateMarkerSpaceVARJO` function is defined as:

```
// Provided by XR_VARJO_marker_tracking
XrResult xrCreateMarkerSpaceVARJO(
    XrSession session,
    const XrMarkerSpaceCreateInfoVARJO* createInfo,
    XrSpace* space);
```

## Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `createInfo` is the structure containing information about how to create the space based on marker.
- `space` is a pointer to a handle in which the created `XrSpace` is returned.

The `xrCreateMarkerSpaceVARJO` function creates marker `XrSpace` for pose relative to the marker specified in `XrMarkerSpaceCreateInfoVARJO`. The runtime **must** return `XR_ERROR_MARKER_ID_INVALID_VARJO` if the supplied `XrMarkerSpaceCreateInfoVARJO::markerId` is invalid.

## Valid Usage (Implicit)

- The `XR_VARJO_marker_tracking` extension **must** be enabled prior to calling `xrCreateMarkerSpaceVARJO`
- `session` **must** be a valid `XrSession` handle
- `createInfo` **must** be a pointer to a valid `XrMarkerSpaceCreateInfoVARJO` structure
- `space` **must** be a pointer to an `XrSpace` handle

## Return Codes

### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_OUT_OF_MEMORY`
- `XR_ERROR_LIMIT_REACHED`
- `XR_ERROR_POSE_INVALID`
- `XR_ERROR_MARKER_ID_INVALID_VARJO`
- `XR_ERROR_FEATURE_UNSUPPORTED`

## New Structures

The `XrSystemMarkerTrackingPropertiesVARJO` structure is defined as:

```
// Provided by XR_VARJO_marker_tracking
typedef struct XrSystemMarkerTrackingPropertiesVARJO {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsMarkerTracking;
} XrSystemMarkerTrackingPropertiesVARJO;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `supportsMarkerTracking` is an [XrBool32](#), indicating if current system is capable of performing marker tracking.

An application **may** inspect whether the system is capable of marker tracking by chaining an [XrSystemMarkerTrackingPropertiesVARJO](#) structure to the [XrSystemProperties](#) structure when calling [xrGetSystemProperties](#).

The runtime **should** return `XR_TRUE` for `supportsMarkerTracking` when marker tracking is available in the system, otherwise `XR_FALSE`. Marker tracking calls **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if marker tracking is not available in the system.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_marker\\_tracking](#) extension **must** be enabled prior to using [XrSystemMarkerTrackingPropertiesVARJO](#)
- `type` **must** be `XR_TYPE_SYSTEM_MARKER_TRACKING_PROPERTIES_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataMarkerTrackingUpdateVARJO](#) structure is defined as:

```
// Provided by XR_VARJO_marker_tracking
typedef struct XrEventDataMarkerTrackingUpdateVARJO {
    XrStructureType    type;
    const void*        next;
    uint64_t           markerId;
    XrBool32           isActive;
    XrBool32           isPredicted;
    XrTime              time;
} XrEventDataMarkerTrackingUpdateVARJO;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `markerId` unique identifier of the marker that has been updated.
- `isActive` the tracking state of the marker.
- `isPredicted` the prediction state of the marker.
- `time` the time of the marker update.

Receiving the [XrEventDataMarkerTrackingUpdateVARJO](#) event structure indicates that the tracking information has changed. The runtime **must** not send more than one event per frame per marker. The runtime **must** send an event if the marker has changed its state (active or inactive). The runtime **must** send an event if it has detected pose change of the active marker.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_marker\\_tracking](#) extension **must** be enabled prior to using [XrEventDataMarkerTrackingUpdateVARJO](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_MARKER_TRACKING_UPDATE_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrMarkerSpaceCreateInfoVARJO](#) structure is defined as:

```
// Provided by XR_VARJO_marker_tracking
typedef struct XrMarkerSpaceCreateInfoVARJO {
    XrStructureType    type;
    const void*        next;
    uint64_t           markerId;
    XrPosef             poseInMarkerSpace;
} XrMarkerSpaceCreateInfoVARJO;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `markerId` unique identifier of the marker.
- `poseInMarkerSpace` is an [XrPosef](#) defining the position and orientation of the new space's origin relative to the marker's natural origin.

## Valid Usage (Implicit)

- The [XR\\_VARJO\\_marker\\_tracking](#) extension **must** be enabled prior to using [XrMarkerSpaceCreateInfoVARJO](#)
- `type` **must** be `XR_TYPE_MARKER_SPACE_CREATE_INFO_VARJO`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SYSTEM_MARKER_TRACKING_PROPERTIES_VARJO`
- `XR_TYPE_EVENT_DATA_MARKER_TRACKING_UPDATE_VARJO`
- `XR_TYPE_MARKER_SPACE_CREATE_INFO_VARJO`

[XrResult](#) enumeration is extended with:

- `XR_ERROR_MARKER_ID_INVALID_VARJO`
- `XR_ERROR_MARKER_NOT_TRACKED_VARJO`

## Issues

## Version History

- Revision 1, 2021-09-30 (Roman Golovanov)

- Initial extension description

## 12.134.2. Example

The example below represents the routine which enables marker tracking feature and then polls events. The event type `XR_TYPE_EVENT_DATA_MARKER_TRACKING_UPDATE_VARJO` has a special handler to process marker state change.

```
1 XrSession session; // previously initialized
2 if(XR_SUCCESS != xrSetMarkerTrackingVARJO(session, XR_TRUE)) {
3     return;
4 }
5
6 XrInstance instance; // previously initialized
7 XrFrameState frameState; // previously initialized
8 XrSpace baseSpace; // previously initialized
9 XrSpaceLocation location; // previously initialized
10
11 // Collection of tracked markers and their space handlers
12 std::unordered_map<uint64_t, XrSpace> markerSpaces;
13 // Initialize an event buffer to hold the output.
14 XrEventDataBuffer event{XR_TYPE_EVENT_DATA_BUFFER};
15 XrResult result = xrPollEvent(instance, &event);
16 if (result == XR_SUCCESS) {
17     switch (event.type) {
18         case XR_TYPE_EVENT_DATA_MARKER_TRACKING_UPDATE_VARJO: {
19             const auto& marker_update =
20                 *reinterpret_cast<XrEventDataMarkerTrackingUpdateVARJO*>(&event);
21
22             const auto id = marker_update.markerId;
23
24             // If marker appeared for the first time then set some settings and
25             // add it to collection
26             if(0 == markerSpaces.count(id)) {
27                 XrMarkerSpaceCreateInfoVARJO spaceInfo
28                 {XR_TYPE_MARKER_SPACE_CREATE_INFO_VARJO};
29                 spaceInfo.markerId = id;
30                 spaceInfo.poseInMarkerSpace = XrPosef{0};
31                 spaceInfo.poseInMarkerSpace.orientation.w = 1.0f;
32                 XrSpace markerSpace;
33                 // Set 1 second timeout
34                 if(XR_SUCCESS != xrSetMarkerTrackingTimeoutVARJO(
35                     session, id, 1000000000))
36                     break;
```



```

37     }
38     // Enable prediction for markers with `odd` ids.
39     if(XR_SUCCESS != xrSetMarkerTrackingPredictionVARJO(
40         session, id, id % 2))
41     {
42         break;
43     }
44     if(XR_SUCCESS != xrCreateMarkerSpaceVARJO(session, &spaceInfo,
45         &markerSpace)) {
46         break;
47     }
48     markerSpaces[id] = markerSpace;
49 }
50
51 if(marker_update.isActive) {
52     if(XR_SUCCESS != xrLocateSpace(markerSpaces.at(id), baseSpace,
53         frameState.predictedDisplayTime, &location)){
54         break;
55     }
56     if(marker_update.isPredicted) {
57         // Process marker as dynamic
58     } else {
59         // Process marker as stationary
60     }
61
62 } else {
63     // Remove previously tracked marker
64     markerSpaces.erase(id);
65 }
66
67 // ...
68 break;
69 }
70 }
71 }

```

## 12.135. XR\_VARJO\_view\_offset

### Name String

XR\_VARJO\_view\_offset

### Extension Type

Instance extension

### Registered Extension Number

126

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-09-30

## IP Status

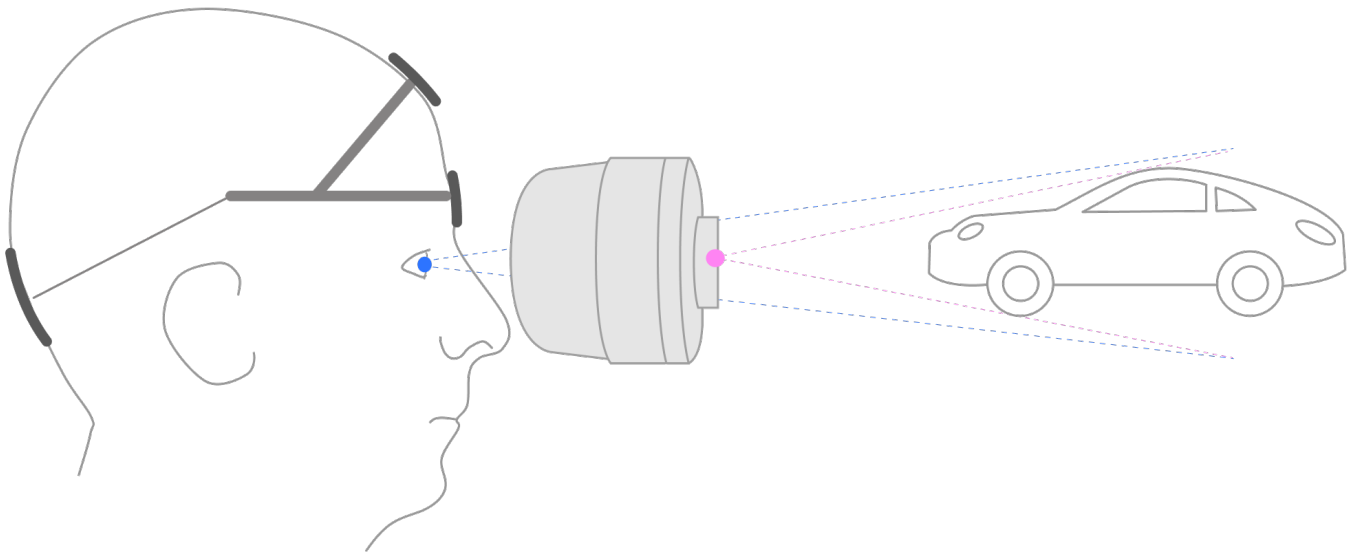
No known IP claims.

## Contributors

Rémi Arnaud, Varjo Technologies

## Overview

Varjo headsets use video pass-through cameras to create the mixed reality (MR) image. The cameras are located around 10 cm (3.9 inches) in front of the user's eyes, which leads to an offset in depth perception so that real-world objects in the video pass-through image appear larger than they are in real life. The image below gives a visualization of the difference between what the camera sees and what the user would see in real life.



This magnification effect is pronounced for objects that are close to the user – for example, their hands

may appear unnaturally large in the image. The effect decreases with distance, so that objects at a distance of 2 meters already appear close to their actual size, and the sizes eventually converge at infinity. Note that while the objects' sizes may differ, their geometry, relative sizes, locations, etc. remain accurate. The extent of the magnification effect ultimately depends both on the application itself and the user's physiology, as the human visual system is highly adaptive in this type of setting.

When blending the video pass-through image with virtual content, it is important that their relative geometries – position, size, and disparity – match one another. To achieve this, Varjo's runtime automatically places the virtual reality cameras in the same position as the physical cameras when the video pass-through feature is enabled (see `XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND`). This allows virtual and real-world content to appear at the same distance and on the same plane when viewed together. While this can be observed as an apparent jump in the location of virtual objects compared to VR-only content, this does not cause any distortion in the object geometry or location; it is only the viewer's location that changes.

In some cases, moving the VR content to match the real-world position may not be desirable. This extension enable the application to control where the VR content is rendered from the location of the user's eyes while the video pass-through image uses the camera locations. For example, if the virtual object is close the user, or if the application is switching between VR and MR modes. Offset values between 0.0 and 1.0 are supported. You can use this to create a smooth, animated transition between the two rendering positions in case you need to change from one to the other during a session.

## New Functions

The `xrSetViewOffsetVARJO` function is defined as:

```
// Provided by XR_VARJO_view_offset
XrResult xrSetViewOffsetVARJO(
    XrSession          session,
    float              offset);
```

### Parameter Descriptions

- `session` is an `XrSession` handle previously created with `xrCreateSession`.
- `offset` is the view offset to be applied. Must be between 0 and 1.

The `xrSetViewOffsetVARJO` function takes a float between 0.0 and 1.0. 0.0 means the pose returned by `xrLocateViews` will be at the eye location, a value of 1.0 means the pose will be at the camera location. A value between 0.0 and 1.0 will interpolate the pose to be in between the eye and the camera location. A value less than 0.0 or more than 1.0 will fail and return error `XR_ERROR_VALIDATION_FAILURE`.

Note that by default the offset is set to 0 if the pass-through cameras are not active, a.k.a. in VR

(`XR_ENVIRONMENT_BLEND_MODE_OPAQUE`), and 1 if the cameras are active, a.k.a. in MR (`XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND` or `XR_ENVIRONMENT_BLEND_MODE_ADDITIVE`).

### Valid Usage (Implicit)

- The `XR_VARJO_view_offset` extension **must** be enabled prior to calling `xrSetViewOffsetVARJO`
- `session` **must** be a valid `XrSession` handle

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`
- `XR_ERROR_FEATURE_UNSUPPORTED`

### Version History

- Revision 1, 2022-02-08 (Remi Arnaud)
  - extension specification

## 12.136. `XR_VARJO_xr4_controller_interaction`

### Name String

`XR_VARJO_xr4_controller_interaction`

### Extension Type

Instance extension

### Registered Extension Number

130

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2023-12-06

## IP Status

No known IP claims.

## Contributors

Denny Rönngren, Varjo Technologies  
Szymon Policht, Varjo Technologies  
Roman Golovanov, Varjo Technologies  
Jussi Karhu, Varjo Technologies

## Overview

This extension adds a new interaction profile for the Varjo Controllers compatible with the Varjo XR-4 headset.

Interaction profile path:

- */interaction\_profiles/varjo/xr-4\_controller*

Valid for the user paths:

- */user/hand/left*
- */user/hand/right*

Supported component paths for */user/hand/left* only:

- *.../input/menu/click*

Supported component paths for */user/hand/right* only:

- *.../input/system/click* (**may** not be available for application use)

Supported component paths on both pathnames:

- *.../input/a/click*
- *.../input/a/touch*
- *.../input/b/click*
- *.../input/b/touch*

- *.../input/squeeze/click*
- *.../input/squeeze/touch*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

## **New Object Types**

## **New Flag Types**

## **New Enum Constants**

## **New Enums**

## **New Structures**

## **New Functions**

## **Issues**

## **Version History**

- Revision 1, 2023-12-06 (Denny Rönngren)
  - Initial extension description

# **12.137. XR\_YVR\_controller\_interaction**

## **Name String**

*XR\_YVR\_controller\_interaction*

## **Extension Type**

Instance extension

## **Registered Extension Number**

498

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

## Last Modified Date

2023-07-12

## IP Status

No known IP claims.

## Contributors

Pengpeng Zhang, YVR  
Xuanyu Chen, YVR

## Overview

This extension defines a new interaction profile for the YVR Controller, including but not limited to YVR1 and YVR2 Controller.

## YVR Controller interaction profile

Interaction profile path:

- */interaction\_profiles/yvr/touch\_controller\_yvr*

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the YVR Controller.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*

- *.../input/y/touch*
- *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
  - *.../input/system/click* (**may** not be available for application use)
- On both:
  - *.../input/squeeze/click*
  - *.../input/trigger/value*
  - *.../input/trigger/touch*
  - *.../input/thumbstick/x*
  - *.../input/thumbstick/y*
  - *.../input/thumbstick/click*
  - *.../input/thumbstick/touch*
  - *.../input/grip/pose*
  - *.../input/aim/pose*
  - *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*





#### *Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

### **New Object Types**

### **New Flag Types**

### **New Enum Constants**

### **New Enums**

### **New Structures**

### **New Functions**

### **Issues**

### **Version History**

- Revision 1, 2023-07-12 (Pengpeng Zhang)
  - Initial extension description

# Chapter 13. List of Provisional Extensions

- [XR\\_EXTX\\_overlay](#)
- [XR\\_HTCX\\_vive\\_tracker\\_interaction](#)
- [XR\\_MNDX\\_egl\\_enable](#)
- [XR\\_MNDX\\_force\\_feedback\\_curl](#)

# 13.1. XR\_EXTX\_overlay

## Name String

XR\_EXTX\_overlay

## Extension Type

Instance extension

## Registered Extension Number

34

## Revision

5

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Last Modified Date

2021-01-13

## IP Status

No known IP claims.

## Contributors

Mark Young, LunarG  
Jules Blok, Epic  
Jared Cheshier, Pluto VR  
Nick Whiting, Epic  
Brad Grantham, LunarG

## Overview

Application developers may desire to implement an OpenXR application that renders content on top of another OpenXR application. These additional applications will execute in a separate process, create a separate session, generate separate content, but want the OpenXR runtime to composite their content on top of the main OpenXR application. Examples of these applications might include:

- A debug environment outputting additional content
- A Store application that hovers to one side of the user's view
- A interactive HUD designed to expose additional chat features

This extension introduces the concept of "Overlay Sessions" in order to expose this usage model.

This extension allows:

- An application to identify when the current sessions composition layers will be applied during composition
- The ability for an overlay session to get information about what is going on with the main application

To enable the functionality of this extension, an application **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo::enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

To create an overlay session, an application **must** pass an `XrSessionCreateInfoOverlayEXTX` structure to `xrCreateSession` via the `XrSessionCreateInfo` structure's `next` parameter.

An overlay application should not assume that the values returned to it by `xrWaitFrame` in `predictedDisplayTime` in `XrFrameState` will be the same as the values returned to the main application or even correlated.

### 13.1.1. Overlay Session Layer Placement

Since one or more sessions may be active at the same time, this extension provides the ability for the application to identify when the frames of the current session will be composited into the final frame.

The `XrSessionCreateInfoOverlayEXTX` `sessionLayersPlacement` parameter provides information on when the sessions composition layers should be applied to the final composition frame. The larger the value passed into `sessionLayersPlacement`, the closer to the front this session's composition layers will appear (relative to other overlay session's composition layers). The smaller the value of `sessionLayersPlacement`, the further to the back this session's composition's layers will appear. The main session's composition layers will always be composited first, resulting in any overlay content being composited on top of the main application's content.

If `sessionLayersPlacement` is 0, then the runtime will always attempt to composite that session's composition layers first. If `sessionLayersPlacement` is `UINT32_MAX`, then the runtime will always attempt to composite that session's composition layers last. If two or more overlay sessions are created with the same `sessionLayersPlacement` value, then the newer session's will be treated as if they had a slightly higher value of `sessionLayersPlacement` than the previous sessions with the same value. This should result in the newest overlay session being composited closer to the user than the older session.

The following image hopefully will provide any further clarification you need:

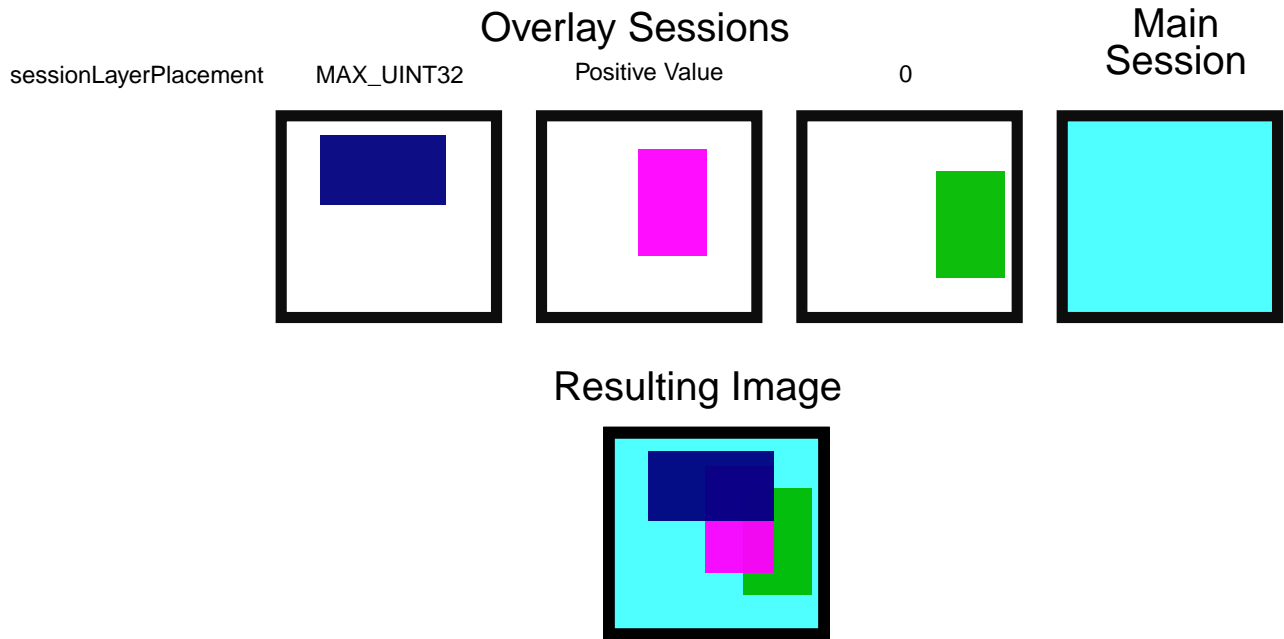


Figure 20. Overlay Composition Order

### 13.1.2. Main Session Behavior Event

Since an overlay session's intends to work in harmony with a main session, some information needs to be provided from that main session to the overlay session.

The [XrEventDataMainSessionVisibilityChangedEXTX](#) event structure provides information on the visibility of the main session as well as some additional flags which can be used to adjust overlay behavior.

If [XR\\_KHR\\_composition\\_layer\\_depth](#) is enabled in the main session, then [XrEventDataMainSessionVisibilityChangedEXTX](#) flags **should** contain the value: [XR\\_OVERLAY\\_MAIN\\_SESSION\\_ENABLED\\_COMPOSITION\\_LAYER\\_INFO\\_DEPTH\\_BIT\\_EXTX](#). If the overlay session also enables [XR\\_KHR\\_composition\\_layer\\_depth](#), then when both sessions are visible, the runtime can integrate their projection layer content together using depth information as described in the extension. However, if either the main session or the overlay do not enable the extension, then composition behavior will continue as if neither one enabled the extension.

### 13.1.3. Modifications to the OpenXR Specification

When this extension is enabled, certain core behaviors defined in the OpenXR specification must change as defined below:

#### Modifications to Composition

The [Compositing](#) section description of the composition process will be changed if this extension is enabled. If this extension is enabled, and there is only one active session, then there is no change. However, if this extension is enabled, and there are multiple active sessions, then the composition will

occur in order based on the overlay session's [XrSessionCreateInfoOverlayEXTX::sessionLayersPlacement](#) value as described in the table below:

Table 6. Overlay Session Composition Order

| Session Type        | <a href="#">XrSessionCreateInfoOverlayEXTX::sessionLayersPlacement</a> | Composited   |
|---------------------|--|--|
| Overlay Session     | UINT32_MAX   | <i>Composited last, appears in front of all other XrSessions</i> |
| Overlay Session     | <Positive value>   |  |
| Overlay Session     | 0  |  |
| Non-overlay Session | N/A  | <i>Composited first, appears behind all other XrSessions</i>     |

The above change only applies to when a session's composition layers are applied to the resulting image. The order in which composition layers are handled internal to a session does not change. However, once the sessions have been properly ordered, the runtime should behave as if all the composition layers have been placed into a single list (maintaining the separation of viewport images) and treat them as if they were from one original session. From this point forward, the composition behavior of the resulting composition layers is the same whether or not this extension is enabled.

If the overlay session is created as part of an [XrInstance](#) which has enabled the [XR\\_KHR\\_composition\\_layer\\_depth](#) extension, and a [XrCompositionLayerDepthInfoKHR](#) structure has been provided to one or more composition layers, then it intends for those layers to be composited into the final image using that depth information. This composition occurs as defined in the [XR\\_KHR\\_composition\\_layer\\_depth](#) extension. However, this is only possible if the main session has provided depth buffer information as part of its swapchain. In the event that a main session does not provide depth buffer information as part of its swapchain, then overlay application's composition layers containing depth information will be composited as if they did not contain that information.

### Modifications to `xrEndFrame` Behavior

[Frame Submission](#) currently states that if `xrEndFrame` is called with no layers, then the runtime should clear the VR display.

If this extension is enabled, the above statement is now only true if the session is not an overlay session. If the session is an overlay session, and it provides 0 layers in the call to `xrEndFrame`, then the runtime will just ignore the overlay session for the current frame.

### Modifications to Input Synchronization

If a runtime supports this extension, it **must** separate input tracking on a per-session basis. This means that reading the input from one active session does not disturb the input information that can be read

by another active session. This may require duplicating events to more than one session.

## New Object Types

None

## New Flag Types

```
typedef XrFlags64 XrOverlayMainSessionFlagsEXTX;
```

```
// Flag bits for XrOverlayMainSessionFlagsEXTX
static const XrOverlayMainSessionFlagsEXTX
XR_OVERLAY_MAIN_SESSION_ENABLED_COMPOSITION_LAYER_INFO_DEPTH_BIT_EXTX = 0x00000001;
```

```
typedef XrFlags64 XrOverlaySessionCreateFlagsEXTX;
```

```
// Flag bits for XrOverlaySessionCreateFlagsEXTX
```

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_SESSION_CREATE_INFO_OVERLAY_EXTX`
- `XR_TYPE_EVENT_DATA_MAIN_SESSION_VISIBILITY_CHANGED_EXTX`

## New Enums

- `XR_OVERLAY_MAIN_SESSION_ENABLED_COMPOSITION_LAYER_INFO_DEPTH_BIT_EXTX`

## New Structures

```
// Provided by XR_EXTX_overlay
typedef struct XrSessionCreateInfoOverlayEXTX {
    XrStructureType          type;
    const void*              next;
    XrOverlaySessionCreateFlagsEXTX createFlags;
    uint32_t                 sessionLayersPlacement;
} XrSessionCreateInfoOverlayEXTX;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `createFlags` is `0` or one or more [XrOverlaySessionCreateFlagBitsEXTX](#) which indicate various characteristics desired for the overlay session.
- `sessionLayersPlacement` is a value indicating the desired placement of the session's composition layers in terms of other sessions.

## Valid Usage (Implicit)

- The [XR\\_EXTX\\_overlay](#) extension **must** be enabled prior to using [XrSessionCreateInfoOverlayEXTX](#)
- `type` **must** be `XR_TYPE_SESSION_CREATE_INFO_OVERLAY_EXTX`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `createFlags` **must** be `0`

```
// Provided by XR_EXTX_overlay
typedef struct XrEventDataMainSessionVisibilityChangedEXTX {
    XrStructureType          type;
    const void*              next;
    XrBool32                 visible;
    XrOverlayMainSessionFlagsEXTX flags;
} XrEventDataMainSessionVisibilityChangedEXTX;
```

Receiving the [XrEventDataMainSessionVisibilityChangedEXTX](#) event structure indicates that the main session has gained or lost visibility. This can occur in many cases, one typical example is when a user switches from one OpenXR application to another. See [XrEventDataMainSessionVisibilityChangedEXTX](#)



for more information on the standard behavior. This structure contains additional information on the main session including `flags` which indicate additional state information of the main session. Currently, the only flag value supplied is `XR_OVERLAY_MAIN_SESSION_ENABLED_COMPOSITION_LAYER_INFO_DEPTH_BIT_EXTX` which indicates if the main session has enabled the `XR_KHR_composition_layer_depth` extension.

### Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `visible` is an `XrBool32` which indicates if the main session is now visible or is not.
- `flags` is 0 or one or more `XrOverlayMainSessionFlagBitsEXTX` which indicates various state information for the main session.

### Valid Usage (Implicit)

- The `XR_EXTX_overlay` extension **must** be enabled prior to using `XrEventDataMainSessionVisibilityChangedEXTX`
- `type` **must** be `XR_TYPE_EVENT_DATA_MAIN_SESSION_VISIBILITY_CHANGED_EXTX`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

### New Functions

None

### New Function Pointers

None

### Issues

None

### Version History

- Revision 1, 2018-11-05 (Mark Young)
  - Initial draft
- Revision 2, 2020-02-12 (Brad Grantham)
  - Name change, remove overlay bool, add flags
- Revision 3, 2020-03-05 (Brad Grantham)

- Name change
- Revision 4, 2020-03-23 (Brad Grantham)
  - Fix enums
- Revision 5, 2021-01-13 (Brad Grantham)
  - Remove bit requesting synchronized display times

## 13.2. XR\_HTCX\_vive\_tracker\_interaction

### Name String

`XR_HTCX_vive_tracker_interaction`

### Extension Type

Instance extension

### Registered Extension Number

104

### Revision

3

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Last Modified Date

2023-07-14

### IP Status

No known IP claims.

### Contributors

Kyle Chen, HTC

Chris Kuo, HTC

### Overview

This extension defines a new interaction profile for HTC VIVE Tracker. HTC VIVE Tracker is a generic tracked device which can be attached to anything to make them trackable. For example, it can be attached to user's hands or feet to track the motion of human body. It can also be attached to any other devices the user wants to track and interact with.

In order to enable the functionality of this extension, you **must** pass the name of the extension into `xrCreateInstance` via the `XrInstanceCreateInfo` `enabledExtensionNames` parameter as indicated in the [Extensions](#) section.

This extension allows:

- An application to enumerate the subpaths of all current connected VIVE trackers.
- An application to receive notification of the top level paths of a VIVE tracker when it is connected.

The paths of a VIVE tracker contains two paths below:

- VIVE tracker persistent path indicate a specific tracker whose lifetime lasts longer than an instance, which means it **must** not change during its hardware lifetime. The format of this path string is unspecified and should be treated as an opaque string.
- VIVE tracker role path **may** be constructed as `"/user/vive_tracker_htcx/role/ROLE_VALUE"`, where `ROLE_VALUE` takes one of the following values. The role path **may** be assigned from the tool provided by the runtime and is `XR_NULL_PATH` if it has not been assigned. If this role path refers to more than one tracker, the runtime **should** choose one of them to be currently active. The role path **may** be changed during the lifetime of instance. Whenever it is changed, the runtime **must** send event `XR_TYPE_EVENT_DATA_VIVE_TRACKER_CONNECTED_HTCX` to provide the new role path of that tracker.

#### ROLE\_VALUE

- `XR_NULL_PATH`
- `handheld_object`
- `left_foot`
- `right_foot`
- `left_shoulder`
- `right_shoulder`
- `left_elbow`
- `right_elbow`
- `left_knee`
- `right_knee`
- `left_wrist` (rev: 3)
- `right_wrist` (rev: 3)
- `left_ankle` (rev: 3)
- `right_ankle` (rev: 3)
- `waist`
- `chest`
- `camera`

- keyboard

- Either the persistent path or the role path can be passed as a subaction path to indicate a specific tracker. For example, `XrActionCreateInfo::subactionPaths` into function `xrCreateAction` or `XrActionSpaceCreateInfo::subactionPath` into function `xrCreateActionSpace`. Please see Example 1 below.

As with other controllers, if a VIVE tracker is connected and bound to a top-level user path, or disconnected while bound to top-level user path, the runtime **must** send event `XR_TYPE_EVENT_DATA_INTERACTION_PROFILE_CHANGED`, and the application **may** call `xrGetCurrentInteractionProfile` to check if the tracker is active or not.



The device that a tracker is attached to probably has a different motion model than what the tracker assumes. The motion tracking might not be as expected in this case.

## VIVE Tracker interaction profile

Interaction profile path:

- `/interaction_profiles/htc/vive_tracker_htcx`

This interaction profile represents the input sources and haptics on the VIVE Tracker.

Supported component paths:

- `.../input/system/click` (**may** not be available for application use)
- `.../input/menu/click`
- `.../input/trigger/click`
- `.../input/squeeze/click`
- `.../input/trigger/value`
- `.../input/trackpad/x`
- `.../input/trackpad/y`
- `.../input/trackpad/click`
- `.../input/trackpad/touch`
- `.../input/grip/pose`
- `.../output/haptic`

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_VIVE\\_TRACKER\\_PATHS\\_HTCX](#)
- [XR\\_TYPE\\_EVENT\\_DATA\\_VIVE\\_TRACKER\\_CONNECTED\\_HTCX](#)

## New Enums

## New Structures

The [XrViveTrackerPathsHTCX](#) structure is defined as:

```
// Provided by XR_HTCX_vive_tracker_interaction
typedef struct XrViveTrackerPathsHTCX {
    XrStructureType    type;
    void*              next;
    XrPath              persistentPath;
    XrPath              rolePath;
} XrViveTrackerPathsHTCX;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `persistentPath` is the unique path of the VIVE tracker which is persistent over the lifetime of the hardware.
- `rolePath` is the path of the VIVE tracker role. This **may** be [XR\\_NULL\\_PATH](#) if the role is not assigned.

The [XrViveTrackerPathsHTCX](#) structure contains two paths of VIVE tracker.

## Valid Usage (Implicit)

- The [XR\\_HTCX\\_vive\\_tracker\\_interaction](#) extension **must** be enabled prior to using [XrViveTrackerPathsHTCX](#)
- `type` **must** be [XR\\_TYPE\\_VIVE\\_TRACKER\\_PATHS\\_HTCX](#)
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrEventDataViveTrackerConnectedHTCX](#) structure is defined as:

```
// Provided by XR_HTCX_vive_tracker_interaction
typedef struct XrEventDataViveTrackerConnectedHTCX {
    XrStructureType      type;
    const void*         next;
    XrViveTrackerPathsHTCX* paths;
} XrEventDataViveTrackerConnectedHTCX;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `paths` contains two paths of the connected VIVE tracker.

Receiving the [XrEventDataViveTrackerConnectedHTCX](#) event structure indicates that a new VIVE tracker was connected or its role changed. It is received via [xrPollEvent](#).

## Valid Usage (Implicit)

- The [XR\\_HTCX\\_vive\\_tracker\\_interaction](#) extension **must** be enabled prior to using [XrEventDataViveTrackerConnectedHTCX](#)
- `type` **must** be `XR_TYPE_EVENT_DATA_VIVE_TRACKER_CONNECTED_HTCX`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

## New Functions

The [xrEnumerateViveTrackerPathsHTCX](#) function is defined as:

```
// Provided by XR_HTCX_vive_tracker_interaction
XrResult xrEnumerateViveTrackerPathsHTCX(
    XrInstance          instance,
    uint32_t            pathCapacityInput,
    uint32_t*           pathCountOutput,
    XrViveTrackerPathsHTCX* paths);
```

## Parameter Descriptions

- `instance` is an instance previously created.
- `pathCapacityInput` is the capacity of the `paths`, or `0` to retrieve the required capacity.
- `pathCountOutput` is a pointer to the count of `XrViveTrackerPathsHTCX` `paths` written, or a pointer to the required capacity in the case that `pathCapacityInput` is insufficient.
- `paths` is a pointer to an array of `XrViveTrackerPathsHTCX` VIVE tracker paths, but **can** be `NULL` if `pathCapacityInput` is `0`.
- See the [Buffer Size Parameters](#) section for a detailed description of retrieving the required `paths` size.

`xrEnumerateViveTrackerPathsHTCX` enumerates all connected VIVE trackers to retrieve their paths under current instance.

## Valid Usage (Implicit)

- The `XR-HTCX-vive-tracker-interaction` extension **must** be enabled prior to calling `xrEnumerateViveTrackerPathsHTCX`
- `instance` **must** be a valid `XrInstance` handle
- `pathCountOutput` **must** be a pointer to a `uint32_t` value
- If `pathCapacityInput` is not `0`, `paths` **must** be a pointer to an array of `pathCapacityInput` `XrViveTrackerPathsHTCX` structures

## Return Codes

### Success

- `XR_SUCCESS`

### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SIZE_INSUFFICIENT`

## Examples

## Example 1

This example illustrates how to locate a VIVE tracker which is attached on the chest. First of all, create an action with `/user/vive_tracker_htcx/role/chest` as the subaction path. Then, submit a suggested binding for that action to the role path plus `.../input/grip/pose`, for the interaction profile `/interaction_profiles/htc/vive_tracker_htcx`, using `xrSuggestInteractionProfileBindings`. To locate the tracker, create an action space from that action, with `/user/vive_tracker_htcx/role/chest` once again specified as the subaction path.

```
extern XrInstance instance; // previously initialized
extern XrSession session; // previously initialized
extern XrActionSet actionSet; // previously initialized

// Create the action with subaction path
XrPath chestTrackerRolePath;
CHK_XR(xrStringToPath(instance, "/user/vive_tracker_htcx/role/chest",
    &chestTrackerRolePath));

XrAction chestPoseAction;
XrActionCreateInfo actionInfo{XR_TYPE_ACTION_CREATE_INFO};
actionInfo.actionType = XR_ACTION_TYPE_POSE_INPUT;
actionInfo.countSubactionPaths = 1;
actionInfo.subactionPaths = &chestTrackerRolePath;
CHK_XR(xrCreateAction(actionSet, &actionInfo, &chestPoseAction));

// Describe a suggested binding for that action and subaction path.
XrPath suggestedBindingPath;
CHK_XR(xrStringToPath(instance,
    "/user/vive_tracker_htcx/role/chest/input/grip/pose",
    &suggestedBindingPath));

std::vector<XrActionSuggestedBinding> actionSuggBindings;
XrActionSuggestedBinding actionSuggBinding;
actionSuggBinding.action = chestPoseAction;
actionSuggBinding.binding = suggestedBindingPath;
actionSuggBindings.push_back(actionSuggBinding);

// Suggest that binding for the VIVE tracker interaction profile
XrPath viveTrackerInteractionProfilePath;
CHK_XR(xrStringToPath(instance, "/interaction_profiles/htc/vive_tracker_htcx",
    &viveTrackerInteractionProfilePath));

XrInteractionProfileSuggestedBinding profileSuggBindings{
    XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING};
profileSuggBindings.interactionProfile =
    viveTrackerInteractionProfilePath;
profileSuggBindings.suggestedBindings =
```



```

    actionSuggBindings.data();
    profileSuggBindings.countSuggestedBindings =
        (uint32_t)actionSuggBindings.size();

    CHK_XR(xrSuggestInteractionProfileBindings(instance, &profileSuggBindings));

    // Create action space for locating tracker
    XrSpace chestTrackerSpace;
    XrActionSpaceCreateInfo actionSpaceInfo{XR_TYPE_ACTION_SPACE_CREATE_INFO};
    actionSpaceInfo.action = chestPoseAction;
    actionSpaceInfo.subactionPath = chestTrackerRolePath;
    CHK_XR(xrCreateActionSpace(session, &actionSpaceInfo, &chestTrackerSpace));

```

## Example 2

This example illustrates how to handle the VIVE tracker when it is connected or disconnected. When a VIVE tracker is connected or its role changed, event `XR_TYPE_EVENT_DATA_VIVE_TRACKER_CONNECTED_HTCX` will be received. The role path and persistent path of this tracker can be retrieved with this event. When a VIVE tracker is connected or disconnected, event `XR_TYPE_EVENT_DATA_INTERACTION_PROFILE_CHANGED` will also be received. The `XrInteractionProfileState::interactionProfile` will be `XR_NULL_PATH` if the tracker represented by that top level path is not connected.

```

extern XrInstance instance; // previously initialized
extern XrSession session; // previously initialized
extern XrEventDataBuffer xrEvent; // previously received from xrPollEvent

switch ( xrEvent.type )
{
    case XR_TYPE_EVENT_DATA_VIVE_TRACKER_CONNECTED_HTCX: {
        const XrEventDataViveTrackerConnectedHTCX& viveTrackerConnected =
            *reinterpret_cast<XrEventDataViveTrackerConnectedHTCX*>(&xrEvent);
        uint32_t nCount;
        char sPersistentPath[XR_MAX_PATH_LENGTH];
        CHK_XR(xrPathToString(instance,
            viveTrackerConnected.paths->persistentPath,
            sizeof(sPersistentPath), &nCount, sPersistentPath));

        std::printf("Vive Tracker connected: %s \n", sPersistentPath);
        if (viveTrackerConnected.paths->rolePath != XR_NULL_PATH) {
            char sRolePath[XR_MAX_PATH_LENGTH];
            CHK_XR(xrPathToString(instance,
                viveTrackerConnected.paths->rolePath, sizeof(sRolePath),
                &nCount, sRolePath));

            std::printf(" New role is: %s\n\n", sRolePath);
        } else {
            std::printf(" No role path.\n\n");
        }
        break;
    }

    case XR_TYPE_EVENT_DATA_INTERACTION_PROFILE_CHANGED: {
        XrPath chestTrackerRolePath;
        XrInteractionProfileState xrInteractionProfileState {
            XR_TYPE_INTERACTION_PROFILE_STATE};

        CHK_XR(xrStringToPath(instance, "/user/vive_tracker_htcx/role/chest",
            &chestTrackerRolePath));
        CHK_XR(xrGetCurrentInteractionProfile(session, chestTrackerRolePath,
            &xrInteractionProfileState));
        break;
    }
}
}

```

## Issues

## Version History

- Revision 1, 2021-09-23 (Kyle Chen)
  - Initial extension description.
- Revision 2, 2022-09-08 (Rylie Pavlik, Collabora, Ltd.)
  - Mark event type as returned-only, updating the implicit valid usage.
- Revision 3, 2022-05-19 (Rune Berg, Valve Corporation)
  - Add new wrist and ankle roles to match additional openvr roles.

## 13.3. XR\_MNDX\_egl\_enable

### Name String

`XR_MNDX_egl_enable`

### Extension Type

Instance extension

### Registered Extension Number

49

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Last Modified Date

2023-12-02

### IP Status

No known IP claims.

### Contributors

Jakob Bornecrantz, Collabora  
Drew DeVault, Individual  
Simon Ser, Individual

### Overview

This extension must be provided by runtimes supporting applications using the EGL API to create rendering contexts.

- [XR\\_USE\\_PLATFORM\\_EGL](#)

### New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- `XR_TYPE_GRAPHICS_BINDING_EGL_MNDX`

## New Enums

## New Structures

The [XrGraphicsBindingEGLMNDX](#) structure is defined as:

```
// Provided by XR_MNDX_egl_enable
typedef struct XrGraphicsBindingEGLMNDX {
    XrStructureType      type;
    const void*         next;
    PFN_xrEglGetProcAddressMNDX  getProcAddress;
    EGLDisplay          display;
    EGLConfig           config;
    EGLContext          context;
} XrGraphicsBindingEGLMNDX;
```

### Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `getProcAddress` is a valid function pointer to [eglGetProcAddress](#).
- `display` is a valid EGL [EGLDisplay](#).
- `config` is a valid EGL [EGLConfig](#).
- `context` is a valid EGL [EGLContext](#).

When creating an EGL based [XrSession](#), the application will provide a pointer to an [XrGraphicsBindingEGLMNDX](#) structure in the `next` chain of the [XrSessionCreateInfo](#).

The required window system configuration define to expose this structure type is [XR\\_USE\\_PLATFORM\\_EGL](#).

## Valid Usage (Implicit)

- The `XR_MNDX_egl_enable` extension **must** be enabled prior to using `XrGraphicsBindingEGLMNDX`
- `type` **must** be `XR_TYPE_GRAPHICS_BINDING_EGL_MNDX`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `getProcAddress` **must** be a valid `PFN_xrEglGetProcAddressMNDX` value
- `display` **must** be a valid `EGLDisplay` value
- `config` **must** be a valid `EGLConfig` value
- `context` **must** be a valid `EGLContext` value

## New Functions

### New Function Pointers

```
typedef PFN_xrVoidFunction (*PFN_xrEglGetProcAddressMNDX)(const char *name);
```

## Parameter Descriptions

- `name` specifies the name of the function to return.

`eglGetProcAddress` returns the address of the client API or EGL function named by `procname`. For details please see <https://registry.khronos.org/EGL/sdk/docs/man/html/eglGetProcAddress.xhtml>

## Issues

### Version History

- Revision 1, 2020-05-20 (Jakob Bornecrantz)
  - Initial draft
- Revision 2, 2023-12-02
  - Use `PFN_xrEglGetProcAddressMNDX` to replace `PFNEGLGETPROCADDRESSPROC` (for `eglGetProcAddress`). Note this does change function pointer attributes on some platforms.

## 13.4. XR\_MNDX\_force\_feedback\_curl

## Name String

`XR_MNDX_force_feedback_curl`

## Extension Type

Instance extension

## Registered Extension Number

376

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)  
and  
[XR\\_EXT\\_hand\\_tracking](#)

## Last Modified Date

2022-11-18

## IP Status

No known IP claims.

## Contributors

Daniel Willmott  
Moses Turner (Collabora, Ltd.)  
Christoph Haagch (Collabora, Ltd.)  
Jakob Bornecrantz (Collabora, Ltd.)

## Overview

This extension provides APIs for force feedback devices capable of restricting physical movement in a single direction along a single dimension.

The intended use for this extension is to provide simple force feedback capabilities to restrict finger movement for VR Gloves.

The application **must** also enable the [XR\\_EXT\\_hand\\_tracking](#) extension in order to use this extension.

The [XrForceFeedbackCurlLocationMNDX](#) describes which location to apply force feedback.

```
// Provided by XR_MNDX_force_feedback_curl
typedef enum XrForceFeedbackCurlLocationMNDX {
    XR_FORCE_FEEDBACK_CURL_LOCATION_THUMB_CURL_MNDX = 0,
    XR_FORCE_FEEDBACK_CURL_LOCATION_INDEX_CURL_MNDX = 1,
    XR_FORCE_FEEDBACK_CURL_LOCATION_MIDDLE_CURL_MNDX = 2,
    XR_FORCE_FEEDBACK_CURL_LOCATION_RING_CURL_MNDX = 3,
    XR_FORCE_FEEDBACK_CURL_LOCATION_LITTLE_CURL_MNDX = 4,
    XR_FORCE_FEEDBACK_CURL_LOCATION_MAX_ENUM_MNDX = 0x7FFFFFFF
} XrForceFeedbackCurlLocationMNDX;
```

## Enumerant Descriptions

- `XR_FORCE_FEEDBACK_CURL_LOCATION_THUMB_CURL_MNDX` — force feedback for thumb curl
- `XR_FORCE_FEEDBACK_CURL_LOCATION_INDEX_CURL_MNDX` — force feedback for index finger curl
- `XR_FORCE_FEEDBACK_CURL_LOCATION_MIDDLE_CURL_MNDX` — force feedback for middle finger curl
- `XR_FORCE_FEEDBACK_CURL_LOCATION_RING_CURL_MNDX` — force feedback for ring finger curl
- `XR_FORCE_FEEDBACK_CURL_LOCATION_LITTLE_CURL_MNDX` — force feedback for little finger curl

### New Object Types

### New Flag Types

### New Enum Constants

`XrStructureType` enumeration is extended with:

- `XR_TYPE_SYSTEM_FORCE_FEEDBACK_CURL_PROPERTIES_MNDX`
- `XR_TYPE_FORCE_FEEDBACK_CURL_APPLY_LOCATIONS_MNDX`

### New Enums

- `XrForceFeedbackCurlLocationMNDX`

### New Structures

The `XrSystemForceFeedbackCurlPropertiesMNDX` structure is defined as:

```
// Provided by XR_MNDX_force_feedback_curl
typedef struct XrSystemForceFeedbackCurlPropertiesMNDX {
    XrStructureType    type;
    void*              next;
    XrBool32           supportsForceFeedbackCurl;
} XrSystemForceFeedbackCurlPropertiesMNDX;
```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `supportsForceFeedbackCurl` is an [XrBool32](#), indicating if the current system is capable of performing force feedback.

An application **may** inspect whether the system is capable of force feedback by chaining an [XrSystemForceFeedbackCurlPropertiesMNDX](#) structure to the [XrSystemProperties](#) structure when calling [xrGetSystemProperties](#).

The runtime **should** return `XR_TRUE` for `supportsForceFeedbackCurl` when force feedback is available in the system, otherwise `XR_FALSE`. Force feedback calls **must** return `XR_ERROR_FEATURE_UNSUPPORTED` if force feedback is not available in the system.

## Valid Usage (Implicit)

- The [XR\\_MNDX\\_force\\_feedback\\_curl](#) extension **must** be enabled prior to using [XrSystemForceFeedbackCurlPropertiesMNDX](#)
- `type` **must** be `XR_TYPE_SYSTEM_FORCE_FEEDBACK_CURL_PROPERTIES_MNDX`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)

The [XrForceFeedbackCurlApplyLocationsMNDX](#) structure is defined as:

```
// Provided by XR_MNDX_force_feedback_curl
typedef struct XrForceFeedbackCurlApplyLocationsMNDX {
    XrStructureType    type;
    const void*        next;
    uint32_t           locationCount;
    XrForceFeedbackCurlApplyLocationMNDX*  locations;
} XrForceFeedbackCurlApplyLocationsMNDX;
```



## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `locationCount` is the number of elements in the `locations` array.
- `locations` is a pointer to an array of locations to apply force feedback.

Contains an array of [XrForceFeedbackCurlApplyLocationMNDX](#) that contains information on locations to apply force feedback to.

## Valid Usage (Implicit)

- The `XR_MNDX_force_feedback_curl` extension **must** be enabled prior to using [XrForceFeedbackCurlApplyLocationsMNDX](#)
- `type` **must** be `XR_TYPE_FORCE_FEEDBACK_CURL_APPLY_LOCATIONS_MNDX`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `locations` **must** be a pointer to an array of `locationCount` [XrForceFeedbackCurlApplyLocationMNDX](#) structures
- The `locationCount` parameter **must** be greater than `0`

The [XrForceFeedbackCurlApplyLocationMNDX](#) structure is defined as:

```
// Provided by XR_MNDX_force_feedback_curl
typedef struct XrForceFeedbackCurlApplyLocationMNDX {
    XrForceFeedbackCurlLocationMNDX    location;
    float                               value;
} XrForceFeedbackCurlApplyLocationMNDX;
```

## Member Descriptions

- **location** represents the location to apply force feedback to.
- **value** is a value from 0-1 representing the amount of force feedback to apply. The range of the value should represent the entire range the location is capable of moving through, with 1 representing making the location incapable of movement, and 0 being fully flexible. For example, in the case of a finger curl, setting **value** to 1 would prevent the finger from curling at all (fully extended), and 0 would allow the finger to have free range of movement, being able to curl fully.

**value** is specified as a limit in a single direction. For example, if the value specified is 0.5, a location **must** have free movement from the point where it would be incapable of movement if **value** was 1, to 0.5 of the range the location is capable of moving.

## Valid Usage (Implicit)

- The `XR_MNDX_force_feedback_curl` extension **must** be enabled prior to using `XrForceFeedbackCurlApplyLocationMNDX`
- **location** **must** be a valid `XrForceFeedbackCurlApplyLocationMNDX` value

## New Functions

The `xrApplyForceFeedbackCurlMNDX` function is defined as:

```
// Provided by XR_MNDX_force_feedback_curl
XrResult xrApplyForceFeedbackCurlMNDX(
    XrHandTrackerEXT          handTracker,
    const XrForceFeedbackCurlApplyLocationsMNDX* locations);
```

## Parameter Descriptions

- **handTracker** is an `XrHandTrackerEXT` handle previously created with `xrCreateHandTrackerEXT`.
- **locations** is an `XrForceFeedbackCurlApplyLocationsMNDX` containing a set of locations to apply force feedback to.

The `xrApplyForceFeedbackCurlMNDX` function applies force feedback to the set locations listed in `XrForceFeedbackCurlApplyLocationsMNDX`.

`xrApplyForceFeedbackCurlMNDX` **should** be called every time an application wishes to update a set of force feedback locations.

Submits a request for force feedback for a set of locations. The runtime **should** deliver this request to the `handTracker` device. If the `handTracker` device is not available, the runtime **may** ignore this request for force feedback.

If the session associated with `handTracker` is not focused, the runtime **must** return `XR_SESSION_NOT_FOCUSED`, and not apply force feedback.

When an application submits force feedback for a set of locations, the runtime **must** update the set of locations to that specified by the application. A runtime **must** set any locations not specified by the application when submitting force feedback to 0.

The runtime **may** discontinue force feedback if the application that set it loses focus. An application **should** call the function again after regaining focus if force feedback is still desired.

### Valid Usage (Implicit)

- The `XR_MNDX_force_feedback_curl` extension **must** be enabled prior to calling `xrApplyForceFeedbackCurlMNDX`
- `handTracker` **must** be a valid `XrHandTrackerEXT` handle
- `locations` **must** be a pointer to a valid `XrForceFeedbackCurlApplyLocationsMNDX` structure

### Return Codes

#### Success

- `XR_SUCCESS`
- `XR_SESSION_LOSS_PENDING`
- `XR_SESSION_NOT_FOCUSED`

#### Failure

- `XR_ERROR_FUNCTION_UNSUPPORTED`
- `XR_ERROR_VALIDATION_FAILURE`
- `XR_ERROR_RUNTIME_FAILURE`
- `XR_ERROR_HANDLE_INVALID`
- `XR_ERROR_INSTANCE_LOST`
- `XR_ERROR_SESSION_LOST`

### Issues

## Version History

- Revision 1, 2022-09-07 (Daniel Willmott)
  - Initial version

# Chapter 14. List of Deprecated Extensions

- `XR_KHR_locate_spaces`
- `XR_KHR_maintenance1`
- `XR_EXT_hp_mixed_reality_controller`
- `XR_EXT_local_floor`
- `XR_EXT_palm_pose`
- `XR_EXT_samsung_odyssey_controller`
- `XR_EXT_uuid`
- `XR_BD_controller_interaction`
- `XR_HTC_vive_cosmos_controller_interaction`
- `XR_HTC_vive_focus3_controller_interaction`
- `XR_ML_m12_controller_interaction`
- `XR_MND_swapchain_usage_input_attachment_bit`
- `XR_OCULUS_android_session_state_enable`
- `XR_VARJO_quad_views`

# 14.1. XR\_KHR\_locate\_spaces

## Name String

`XR_KHR_locate_spaces`

## Extension Type

Instance extension

## Registered Extension Number

472

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Deprecation State

- *Promoted to* [OpenXR 1.1](#)

## Last Modified Date

2024-01-19

## IP Status

No known IP claims.

## Contributors

Yin Li, Microsoft  
Bryce Hutchings, Microsoft  
Andreas Loeve Selvik, Meta Platforms  
John Kearney, Meta Platforms  
Robert Blenkinsopp, Ultraleap  
Rylie Pavlik, Collabora  
Ron Bessems, Magic Leap  
Jakob Bornecrantz, NVIDIA

### 14.1.1. Overview

This extension introduces the [xrLocateSpacesKHR](#) function, which enables applications to locate an array of spaces in a single function call. Runtimes **may** provide performance benefits for applications that use many spaces.

Compared to the [xrLocateSpace](#) function, the new [xrLocateSpacesKHR](#) function also provides extensible input parameters for future extensions to extend using additional chained structures.

## 14.1.2. Locate spaces

Applications **can** use [xrLocateSpacesKHR](#) function to locate an array of spaces.

The [xrLocateSpacesKHR](#) function is defined as:

```
// Provided by XR_KHR_locate_spaces
XrResult xrLocateSpacesKHR(
    XrSession          session,
    const XrSpacesLocateInfo* locateInfo,
    XrSpaceLocations* spaceLocations);
```

### Parameter Descriptions

- **session** is an [XrSession](#) handle previously created with [xrCreateSession](#).
- **locateInfo** is a pointer to an [XrSpacesLocateInfoKHR](#) that provides the input information to locate spaces.
- **spaceLocations** is a pointer to an [XrSpaceLocationsKHR](#) for the runtime to return the locations of the specified spaces in the base space.

[xrLocateSpacesKHR](#) provides the physical location of one or more spaces in a base space at a specified time, if currently known by the runtime.

The [XrSpacesLocateInfoKHR::time](#), the [XrSpacesLocateInfoKHR::baseSpace](#), and each space in [XrSpacesLocateInfoKHR::spaces](#), in the **locateInfo** parameter, all follow the same specifics as the corresponding inputs to the [xrLocateSpace](#) function.

### Valid Usage (Implicit)

- The [XR\\_KHR\\_locate\\_spaces](#) extension **must** be enabled prior to calling [xrLocateSpacesKHR](#)
- **session** **must** be a valid [XrSession](#) handle
- **locateInfo** **must** be a pointer to a valid [XrSpacesLocateInfo](#) structure
- **spaceLocations** **must** be a pointer to an [XrSpaceLocations](#) structure

## Return Codes

### Success

- XR\_SUCCESS
- XR\_SESSION\_LOSS\_PENDING

### Failure

- XR\_ERROR\_FUNCTION\_UNSUPPORTED
- XR\_ERROR\_VALIDATION\_FAILURE
- XR\_ERROR\_RUNTIME\_FAILURE
- XR\_ERROR\_HANDLE\_INVALID
- XR\_ERROR\_INSTANCE\_LOST
- XR\_ERROR\_SESSION\_LOST
- XR\_ERROR\_SIZE\_INSUFFICIENT
- XR\_ERROR\_TIME\_INVALID

The [XrSpacesLocateInfoKHR](#) structure is defined as:

```
// Provided by XR_KHR_locate_spaces
// XrSpacesLocateInfoKHR is an alias for XrSpacesLocateInfo
typedef struct XrSpacesLocateInfo {
    XrStructureType    type;
    const void*        next;
    XrSpace             baseSpace;
    XrTime              time;
    uint32_t           spaceCount;
    const XrSpace*     spaces;
} XrSpacesLocateInfo;

typedef XrSpacesLocateInfo XrSpacesLocateInfoKHR;
```



## Member Descriptions

- `type` is the `XrStructureType` of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain. No such structures are defined in core OpenXR or this extension.
- `baseSpace` identifies the underlying space in which to locate `spaces`.
- `time` is the time for which the location is requested.
- `spaceCount` is a `uint32_t` specifying the count of elements in the `spaces` array.
- `spaces` is an array of valid `XrSpace` handles to be located.

The `time`, the `baseSpace`, and each space in `spaces` all follow the same specifics as the corresponding inputs to the `xrLocateSpace` function.

The `baseSpace` and all of the `XrSpace` handles in the `spaces` array **must** be valid and share the same parent `XrSession`.

If the `time` is invalid, the `xrLocateSpacesKHR` **must** return `XR_ERROR_TIME_INVALID`.

The `spaceCount` **must** be a positive number, i.e. the array `spaces` **must** not be empty. Otherwise, the runtime **must** return `XR_ERROR_VALIDATION_FAILURE`.

## Valid Usage (Implicit)

- The `XR_KHR_locate_spaces` extension **must** be enabled prior to using `XrSpacesLocateInfoKHR`
- **Note:** `XrSpacesLocateInfoKHR` is an alias for `XrSpacesLocateInfo`, so the following items replicate the implicit valid usage for `XrSpacesLocateInfo`
- `type` **must** be `XR_TYPE_SPACES_LOCATE_INFO`
- `next` **must** be `NULL` or a valid pointer to the `next structure in a structure chain`
- `baseSpace` **must** be a valid `XrSpace` handle
- `spaces` **must** be a pointer to an array of `spaceCount` valid `XrSpace` handles
- The `spaceCount` parameter **must** be greater than `0`
- Both of `baseSpace` and the elements of `spaces` **must** have been created, allocated, or retrieved from the same `XrSession`

The `XrSpaceLocationsKHR` structure is defined as:

```

// Provided by XR_KHR_locate_spaces
// XrSpaceLocationsKHR is an alias for XrSpaceLocations
typedef struct XrSpaceLocations {
    XrStructureType    type;
    void*              next;
    uint32_t           locationCount;
    XrSpaceLocationData* locations;
} XrSpaceLocations;

typedef XrSpaceLocations XrSpaceLocationsKHR;

```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain, such as [XrSpaceVelocitiesKHR](#).
- `locationCount` is a `uint32_t` specifying the count of elements in the `locations` array.
- `locations` is an array of [XrSpaceLocationsKHR](#) for the runtime to populate with the locations of the specified spaces in the [XrSpacesLocateInfoKHR::baseSpace](#) at the specified [XrSpacesLocateInfoKHR::time](#).

The [XrSpaceLocationsKHR](#) structure contains an array of space locations in the member `locations`, to be used as output for [xrLocateSpacesKHR](#). The application **must** allocate this array to be populated with the function output. The `locationCount` value **must** be the same as [XrSpacesLocateInfoKHR::spaceCount](#), otherwise, the [xrLocateSpacesKHR](#) function **must** return `XR_ERROR_VALIDATION_FAILURE`.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_locate\\_spaces](#) extension **must** be enabled prior to using [XrSpaceLocationsKHR](#)
- **Note:** [XrSpaceLocationsKHR](#) is an alias for [XrSpaceLocations](#), so the following items replicate the implicit valid usage for [XrSpaceLocations](#)
- `type` **must** be `XR_TYPE_SPACE_LOCATIONS`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#). See also: [XrSpaceVelocities](#)
- `locations` **must** be a pointer to an array of `locationCount` [XrSpaceLocationData](#) structures
- The `locationCount` parameter **must** be greater than `0`

The [XrSpaceLocationDataKHR](#) structure is defined as:

```

// Provided by XR_KHR_locate_spaces
// XrSpaceLocationDataKHR is an alias for XrSpaceLocationData
typedef struct XrSpaceLocationData {
    XrSpaceLocationFlags    locationFlags;
    XrPosef                 pose;
} XrSpaceLocationData;

typedef XrSpaceLocationData XrSpaceLocationDataKHR;

```

### Member Descriptions

- `locationFlags` is a bitfield, with bit masks defined in [XrSpaceLocationFlagBits](#). It behaves the same as [XrSpaceLocation::locationFlags](#).
- `pose` is an [XrPosef](#) that behaves the same as [XrSpaceLocation::pose](#).

This is a single element of the array in [XrSpaceLocationsKHR::locations](#), and is used to return the pose and location flags for a single space with respect to the specified base space from a call to [xrLocateSpacesKHR](#). It does not accept chained structures to allow for easier use in dynamically allocated container datatypes. Chained structures are possible with the [XrSpaceLocationsKHR](#) that describes an array of these elements.

### Valid Usage (Implicit)

- The [XR\\_KHR\\_locate\\_spaces](#) extension **must** be enabled prior to using [XrSpaceLocationDataKHR](#)

### 14.1.3. Locate space velocities

Applications **can** request the velocities of spaces by chaining the [XrSpaceVelocitiesKHR](#) structure to the next pointer of [XrSpaceLocationsKHR](#) when calling [xrLocateSpacesKHR](#).

The [XrSpaceVelocitiesKHR](#) structure is defined as:

```

// Provided by XR_KHR_locate_spaces
// XrSpaceVelocitiesKHR is an alias for XrSpaceVelocities
typedef struct XrSpaceVelocities {
    XrStructureType    type;
    void*              next;
    uint32_t           velocityCount;
    XrSpaceVelocityData* velocities;
} XrSpaceVelocities;

typedef XrSpaceVelocities XrSpaceVelocitiesKHR;

```

## Member Descriptions

- `type` is the [XrStructureType](#) of this structure.
- `next` is `NULL` or a pointer to the next structure in a structure chain.
- `velocityCount` is a `uint32_t` specifying the count of elements in the `velocities` array.
- `velocities` is an array of [XrSpaceVelocityDataKHR](#) for the runtime to populate with the velocities of the specified spaces in the [XrSpacesLocateInfoKHR::baseSpace](#) at the specified [XrSpacesLocateInfoKHR::time](#).

The `velocities` member contains an array of space velocities in the member `velocities`, to be used as output for [xrLocateSpacesKHR](#). The application **must** allocate this array to be populated with the function output. The `velocityCount` value **must** be the same as [XrSpacesLocateInfoKHR::spaceCount](#), otherwise, the [xrLocateSpacesKHR](#) function **must** return `XR_ERROR_VALIDATION_FAILURE`.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_locate\\_spaces](#) extension **must** be enabled prior to using [XrSpaceVelocitiesKHR](#)
- **Note:** [XrSpaceVelocitiesKHR](#) is an alias for [XrSpaceVelocities](#), so the following items replicate the implicit valid usage for [XrSpaceVelocities](#)
- `type` **must** be `XR_TYPE_SPACE_VELOCITIES`
- `next` **must** be `NULL` or a valid pointer to the [next structure in a structure chain](#)
- `velocities` **must** be a pointer to an array of `velocityCount` [XrSpaceVelocityData](#) structures
- The `velocityCount` parameter **must** be greater than `0`

The [XrSpaceVelocityDataKHR](#) structure is defined as:

```

// Provided by XR_KHR_locate_spaces
// XrSpaceVelocityDataKHR is an alias for XrSpaceVelocityData
typedef struct XrSpaceVelocityData {
    XrSpaceVelocityFlags    velocityFlags;
    XrVector3f              linearVelocity;
    XrVector3f              angularVelocity;
} XrSpaceVelocityData;

typedef XrSpaceVelocityData XrSpaceVelocityDataKHR;

```

## Member Descriptions

- `velocityFlags` is a bitfield, with bit values defined in [XrSpaceVelocityFlagBits](#). It behaves the same as [XrSpaceVelocity::velocityFlags](#).
- `linearVelocity` is an [XrVector3f](#). It behaves the same as [XrSpaceVelocity::linearVelocity](#).
- `angularVelocity` is an [XrVector3f](#). It behaves the same as [XrSpaceVelocity::angularVelocity](#).

This is a single element of the array in [XrSpaceVelocitiesKHR::velocities](#), and is used to return the linear and angular velocity and velocity flags for a single space with respect to the specified base space from a call to [xrLocateSpacesKHR](#). It does not accept chained structures to allow for easier use in dynamically allocated container datatypes.

## Valid Usage (Implicit)

- The [XR\\_KHR\\_locate\\_spaces](#) extension **must** be enabled prior to using [XrSpaceVelocityDataKHR](#)

### 14.1.4. Example code for [xrLocateSpacesKHR](#)

The following example code shows how an application retrieves both the location and velocity of one or more spaces in a base space at a given time using the [xrLocateSpacesKHR](#) function.

```

XrInstance instance; // previously initialized
XrSession session; // previously initialized
XrSpace baseSpace; // previously initialized
std::vector<XrSpace> spacesToLocate; // previously initialized

// Prepare output buffers to receive data and get reused in frame loop.
std::vector<XrSpaceLocationDataKHR> locationBuffer(spacesToLocate.size());
std::vector<XrSpaceVelocityDataKHR> velocityBuffer(spacesToLocate.size());

// Get function pointer for xrLocateSpacesKHR.

```

```

PFN_xrLocateSpacesKHR xrLocateSpacesKHR;
CHK_XR(xrGetInstanceProcAddr(instance, "xrLocateSpacesKHR",
                             reinterpret_cast<PFN_xrVoidFunction*>(
                                 &xrLocateSpacesKHR)));

// application frame loop
while (1) {
    // Typically the time is the predicted display time returned from xrWaitFrame.
    XrTime displayTime; // previously initialized.

    XrSpacesLocateInfoKHR locateInfo{XR_TYPE_SPACES_LOCATE_INFO_KHR};
    locateInfo.baseSpace = baseSpace;
    locateInfo.time = displayTime;
    locateInfo.spaceCount = (uint32_t)spacesToLocate.size();
    locateInfo.spaces = spacesToLocate.data();

    XrSpaceLocationsKHR locations{XR_TYPE_SPACES_LOCATE_INFO_KHR};
    locations.locationCount = (uint32_t)locationBuffer.size();
    locations.locations = locationBuffer.data();

    XrSpaceVelocitiesKHR velocities{XR_TYPE_SPACE_VELOCITIES_KHR};
    velocities.velocityCount = (uint32_t)velocityBuffer.size();
    velocities.velocities = velocityBuffer.data();

    locations.next = &velocities;
    CHK_XR(xrLocateSpacesKHR(session, &locateInfo, &locations));

    for (uint32_t i = 0; i < spacesToLocate.size(); i++) {
        const auto positionAndOrientationTracked =
            XR_SPACE_LOCATION_POSITION_TRACKED_BIT |
XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT;
        const auto orientationOnlyTracked = XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT;

        if ((locationBuffer[i].locationFlags & positionAndOrientationTracked) ==
positionAndOrientationTracked) {
            // if the location is 6dof tracked
            do_something(locationBuffer[i].pose.position);
            do_something(locationBuffer[i].pose.orientation);

            const auto velocityValidBits =
                XR_SPACE_VELOCITY_LINEAR_VALID_BIT | XR_SPACE_VELOCITY_ANGULAR_VALID_BIT;
            if ((velocityBuffer[i].velocityFlags & velocityValidBits) ==
velocityValidBits) {
                do_something(velocityBuffer[i].linearVelocity);
                do_something(velocityBuffer[i].angularVelocity);
            }
        }
        else if ((locationBuffer[i].locationFlags & orientationOnlyTracked) ==

```

```
orientationOnlyTracked) {
    // if the location is 3dof tracked
    do_something(locationBuffer[i].pose.orientation);

    if ((velocityBuffer[i].velocityFlags & XR_SPACE_VELOCITY_ANGULAR_VALID_BIT)
== XR_SPACE_VELOCITY_ANGULAR_VALID_BIT) {
        do_something(velocityBuffer[i].angularVelocity);
    }
}
}
```

## New Object Types

## New Flag Types

## New Enum Constants

[XrStructureType](#) enumeration is extended with:

- [XR\\_TYPE\\_SPACES\\_LOCATE\\_INFO\\_KHR](#)
- [XR\\_TYPE\\_SPACE\\_LOCATIONS\\_KHR](#)
- [XR\\_TYPE\\_SPACE\\_VELOCITIES\\_KHR](#)

## New Enums

## New Structures

- [XrSpacesLocateInfoKHR](#)
- [XrSpaceLocationsKHR](#)
- [XrSpaceLocationDataKHR](#)
- [XrSpaceVelocitiesKHR](#)
- [XrSpaceVelocityDataKHR](#)

## New Functions

- [xrLocateSpacesKHR](#)

## Issues

## Version History

- Revision 1, 2023-04-22 (Yin LI)
  - Initial extension description

## 14.2. XR\_KHR\_maintenance1

### Name String

`XR_KHR_maintenance1`

### Extension Type

Instance extension

### Registered Extension Number

711

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_BD\\_controller\\_interaction](#)
- Interacts with [XR\\_EXT\\_hp\\_mixed\\_reality\\_controller](#)
- Interacts with [XR\\_EXT\\_samsung\\_odyssey\\_controller](#)
- Interacts with [XR\\_FB\\_touch\\_controller\\_pro](#)
- Interacts with [XR\\_HTCX\\_vive\\_tracker\\_interaction](#)
- Interacts with [XR\\_HTC\\_hand\\_interaction](#)
- Interacts with [XR\\_HTC\\_vive\\_cosmos\\_controller\\_interaction](#)
- Interacts with [XR\\_HTC\\_vive\\_focus3\\_controller\\_interaction](#)
- Interacts with [XR\\_HUAWEI\\_controller\\_interaction](#)
- Interacts with [XR\\_META\\_touch\\_controller\\_plus](#)
- Interacts with [XR\\_ML\\_m12\\_controller\\_interaction](#)
- Interacts with [XR\\_MSFT\\_hand\\_interaction](#)
- Interacts with [XR\\_OPPO\\_controller\\_interaction](#)
- Interacts with [XR\\_YVR\\_controller\\_interaction](#)

### Deprecation State

- *Promoted* to [OpenXR 1.1](#)

### Last Modified Date

2023-10-25



## IP Status

No known IP claims.

## Contributors

Ron Bessems, Magic Leap  
Karthik Kadappan, Magic Leap  
Rylie Pavlik, Collabora  
Nihav Jain, Google  
Lachlan Ford, Google  
John Kearney, Meta  
Yin Li, Microsoft  
Robert Blenkinsopp, Ultraleap

### 14.2.1. Overview

[XR\\_KHR\\_maintenance1](#) adds a collection of minor features that were intentionally left out or overlooked from the original OpenXR 1.0 release. All are promoted to the OpenXR 1.1 release.

```
// Provided by XR_KHR_maintenance1
// XrColor3fKHR is an alias for XrColor3f
typedef struct XrColor3f {
    float    r;
    float    g;
    float    b;
} XrColor3f;

typedef XrColor3f XrColor3fKHR;
```

```
// Provided by XR_KHR_maintenance1
// XrExtent3DfKHR is an alias for XrExtent3Df
typedef struct XrExtent3Df {
    float    width;
    float    height;
    float    depth;
} XrExtent3Df;

typedef XrExtent3Df XrExtent3DfKHR;
```

```
// Provided by XR_KHR_maintenance1
// XrSpherefKHR is an alias for XrSpheref
typedef struct XrSpheref {
    XrPosef    center;
    float      radius;
} XrSpheref;

typedef XrSpheref XrSpherefKHR;
```

```
// Provided by XR_KHR_maintenance1
// XrBoxfKHR is an alias for XrBoxf
typedef struct XrBoxf {
    XrPosef    center;
    XrExtent3Df extents;
} XrBoxf;

typedef XrBoxf XrBoxfKHR;
```

```
// Provided by XR_KHR_maintenance1
// XrFrustumfKHR is an alias for XrFrustumf
typedef struct XrFrustumf {
    XrPosef    pose;
    XrFovf     fov;
    float      nearZ;
    float      farZ;
} XrFrustumf;

typedef XrFrustumf XrFrustumfKHR;
```

### 14.2.2. New Structures

- [XrBoxfKHR](#)
- [XrColor3fKHR](#)
- [XrExtent3DfKHR](#)
- [XrFrustumfKHR](#)
- [XrSpherefKHR](#)

### 14.2.3. New Enum Constants

- `XR_KHR_MAINTENANCE1_EXTENSION_NAME`
- `XR_KHR_maintenance1_SPEC_VERSION`
- Extending [XrResult](#):
  - `XR_ERROR_EXTENSION_DEPENDENCY_NOT_ENABLED_KHR`
  - `XR_ERROR_PERMISSION_INSUFFICIENT_KHR`

### 14.2.4. Version History

- Revision 1, 2023-10-25 (Ron Bessems)
  - Initial extension description

## 14.3. XR\_EXT\_hp\_mixed\_reality\_controller

### Name String

`XR_EXT_hp_mixed_reality_controller`

### Extension Type

Instance extension

### Registered Extension Number

96

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Deprecation State

- *Promoted* to [OpenXR 1.1](#)

### Last Modified Date

2020-06-08

## IP Status

No known IP claims.

## Contributors

Alain Zanchetta, Microsoft  
Lachlan Ford, Microsoft  
Alex Turner, Microsoft  
Yin Li, Microsoft  
Nathan Nuber, HP Inc.

## Overview

This extension added a new interaction profile path for the HP Reverb G2 Controllers:

- */interaction\_profiles/hp/mixed\_reality\_controller*

### Note

The interaction profile path */interaction\_profiles/hp/mixed\_reality\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/hp/mixed\_reality\_controller\_hp*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for the user paths

- */user/hand/left*
- */user/hand/right*

Supported component paths:

- On */user/hand/left* only
  - *.../input/x/click*
  - *.../input/y/click*
- On */user/hand/right* only
  - *.../input/a/click*
  - *.../input/b/click*
- On both hands
  - *.../input/menu/click*
  - *.../input/squeeze/value*
  - *.../input/trigger/value*
  - *.../input/thumbstick/x*

- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## Version History

- Revision 1, 2020-06-08 (Yin Li)
  - Initial extension proposal

## 14.4. XR\_EXT\_local\_floor

### Name String

`XR_EXT_local_floor`

## Extension Type

Instance extension

## Registered Extension Number

427

## Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Deprecation State

- *Promoted to* [OpenXR 1.1](#)

## Last Modified Date

2022-11-28

## IP Status

No known IP claims.

## Contributors

John Kearney, Meta

Alex Turner, Microsoft

Yin Li, Microsoft

Cass Everitt, Meta

## Contacts

John Kearney, Meta

## Overview

The core OpenXR spec contains two world-locked reference space [XrSpace](#) types in [XrReferenceSpaceType](#), `XR_REFERENCE_SPACE_TYPE_LOCAL` and `XR_REFERENCE_SPACE_TYPE_STAGE` with a design goal that `LOCAL` space gets the user positioned correctly in `XZ` space and `STAGE` gets the user positioned correctly in `Y` space.

As defined in the core OpenXR spec, `LOCAL` space is useful when an application needs to render **seated-scale** content that is not positioned relative to the physical floor and `STAGE` space is useful when an application needs to render **standing-scale** content that is relative to the physical floor.

The core OpenXR specification describes that **standing-scale** experiences are meant to use the `STAGE` reference space. However, using the `STAGE` forces the user to move to the stage space in order to operate their experience, rather than just standing locally where they are.

## Definition of the space

Similar to `LOCAL` space, the `LOCAL_FLOOR` reference space (`XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR_EXT`) establishes a world-locked origin, gravity-aligned to exclude pitch and roll, with +Y up, +X to the right, and -Z forward.

The location of the origin of the `LOCAL_FLOOR` space **must** match the `LOCAL` space in the X and Z coordinates but not in the Y coordinate.

The orientation of the `LOCAL_FLOOR` space **must** match the `LOCAL` space.

If the `STAGE` space is supported, then the floor level (Y coordinate) of the `LOCAL_FLOOR` space and the `STAGE` space **must** match.

If the `STAGE` space is not supported, then the runtime **must** give a best estimate of the floor level.

Note: The `LOCAL_FLOOR` space could be implemented by an application without support from the runtime by using the difference between in the Y coordinate of the pose of the `LOCAL` and `STAGE` reference spaces.

When this extension is enabled, a runtime **must** support `XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR_EXT` (in [xrEnumerateReferenceSpaces](#)).

When a user needs to recenter `LOCAL` space, the `LOCAL_FLOOR` space will also be recentered.

When such a recentering occurs, the runtime **must** queue the [XrEventDataReferenceSpaceChangePending](#) event, with the recentered `LOCAL_FLOOR` space origin only taking effect for [xrLocateSpace](#) or [xrLocateViews](#) calls whose `XrTime` parameter is greater than or equal to the `changeTime` provided in that event. Additionally, when the runtime changes the floor level (or the floor level estimate), the runtime **must** queue this event.

## New Object Types

## New Flag Types

## New Enum Constants

[XrReferenceSpaceType](#) enumeration is extended with:

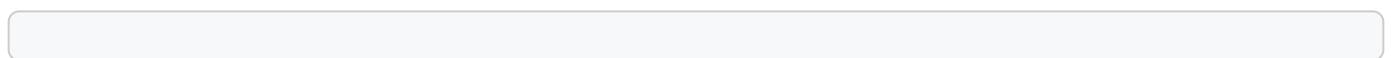
- `XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR_EXT`

## New Enums

## New Structures

## Examples

If a runtime does not support the local floor extension, an application **can** construct an equivalent space using the `LOCAL` and `STAGE` spaces.



```

extern XrSession session;
extern bool supportsStageSpace;
extern bool supportsLocalFloorExtension;
extern XrTime curtime; // previously initialized

XrSpace localFloorSpace = XR_NULL_HANDLE;

if (supportsLocalFloorExtension)
{
    XrReferenceSpaceCreateInfo localFloorCreateInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};
    localFloorCreateInfo.poseInReferenceSpace = {{0.f, 0.f, 0.f, 1.f}, {0.f, 0.f, 0.f}};
    localFloorCreateInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_LOCAL_FLOOR_EXT;
    CHK_XR(xrCreateReferenceSpace(session, &localFloorCreateInfo, &localFloorSpace));
}
else if (supportsStageSpace)
{
    XrSpace localSpace = XR_NULL_HANDLE;
    XrSpace stageSpace = XR_NULL_HANDLE;

    XrReferenceSpaceCreateInfo createInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};
    createInfo.poseInReferenceSpace.orientation.w = 1.f;

    createInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_LOCAL;
    CHK_XR(xrCreateReferenceSpace(session, &createInfo, &localSpace));

    createInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_STAGE;
    CHK_XR(xrCreateReferenceSpace(session, &createInfo, &stageSpace));

    XrSpaceLocation stageLoc{XR_TYPE_SPACE_LOCATION};
    CHK_XR(xrLocateSpace(stageSpace, localSpace, curtime, &stageLoc));

    CHK_XR(xrDestroySpace(localSpace));
    CHK_XR(xrDestroySpace(stageSpace));

    float floorOffset = stageLoc.pose.position.y;

    XrReferenceSpaceCreateInfo localFloorCreateInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};
    localFloorCreateInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_LOCAL;
    localFloorCreateInfo.poseInReferenceSpace = {{0.f, 0.f, 0.f, 1.f}, {0.f, floorOffset,
0.f}};
    CHK_XR(xrCreateReferenceSpace(session, &localFloorCreateInfo, &localFloorSpace));
}
else
{
    // We do not support local floor or stage - make an educated guess
    float floorOffset = -1.5;

    XrReferenceSpaceCreateInfo localFloorCreateInfo{XR_TYPE_REFERENCE_SPACE_CREATE_INFO};

```



```
localFloorCreateInfo.referenceSpaceType = XR_REFERENCE_SPACE_TYPE_LOCAL;
localFloorCreateInfo.poseInReferenceSpace = {{0.f, 0.f, 0.f, 1.f}, {0.f, floorOffset,
0.f}}};
CHK_XR(xrCreateReferenceSpace(session, &localFloorCreateInfo, &localFloorSpace));
}
```

## Issues

None

## Version History

- Revision 1, 2022-11-28 (John Kearney)
  - Initial draft

# 14.5. XR\_EXT\_palm\_pose

## Name String

`XR_EXT_palm_pose`

## Extension Type

Instance extension

## Registered Extension Number

177

## Revision

3

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Deprecation State

- *Promoted to* [OpenXR 1.1](#)

## Last Modified Date

2022-05-23

## IP Status

No known IP claims.

## Contributors

Jack Pritz, Unity Technologies  
Joe Ludwig, Valve  
Rune Berg, Valve

John Kearney, Facebook  
Peter Kuhn, Unity Technologies  
Lachlan Ford, Microsoft

## Overview

This extension defines a new "standard pose identifier" for interaction profiles, named "palm\_ext". The new identifier is a pose that can be used to place application-specific visual content such as avatar visuals that may or may not match human hands. This extension also adds a new input component path using this "palm\_ext" pose identifier to existing interaction profiles when active.

The application **can** use the `.../input/palm_ext/pose` component path to place visual content representing the user's physical hand location. Application visuals may depict, for example, realistic human hands that are very simply animated or creative depictions such as an animal, an alien, or robot limb extremity.

Note that this is not intended to be an alternative to extensions that perform hand tracking for more complex use cases: the use of "palm" in the name is to reflect that it is a user-focused pose rather than a held-object-focused pose.



### Note

OpenXR 1.1 replaces `.../input/palm_ext/pose` with `.../input/grip_surface/pose`. The definitions of both poses are identical.

## Pose Identifier

When this extension is active, a runtime **must** behave as if the following were added to the list of [Standard pose identifiers](#).

- palm\_ext - a pose that allows applications to reliably anchor visual content relative to the user's physical hand, whether the user's hand is tracked directly or its position and orientation is inferred by a physical controller. The palm pose is defined as follows:
  - The palm position: The user's physical palm centroid, at the surface of the palm.
  - The palm orientation's +X axis: When a user is holding the controller and straightens their index finger, the ray that is normal to the user's palm (away from the palm in the left hand, into the palm in the right hand).
  - The palm orientation's -Z axis: When a user is holding the controller and straightens their index finger, the ray that is parallel to their finger's pointing direction.
  - The palm orientation's +Y axis: orthogonal to +Z and +X using the right-hand rule.

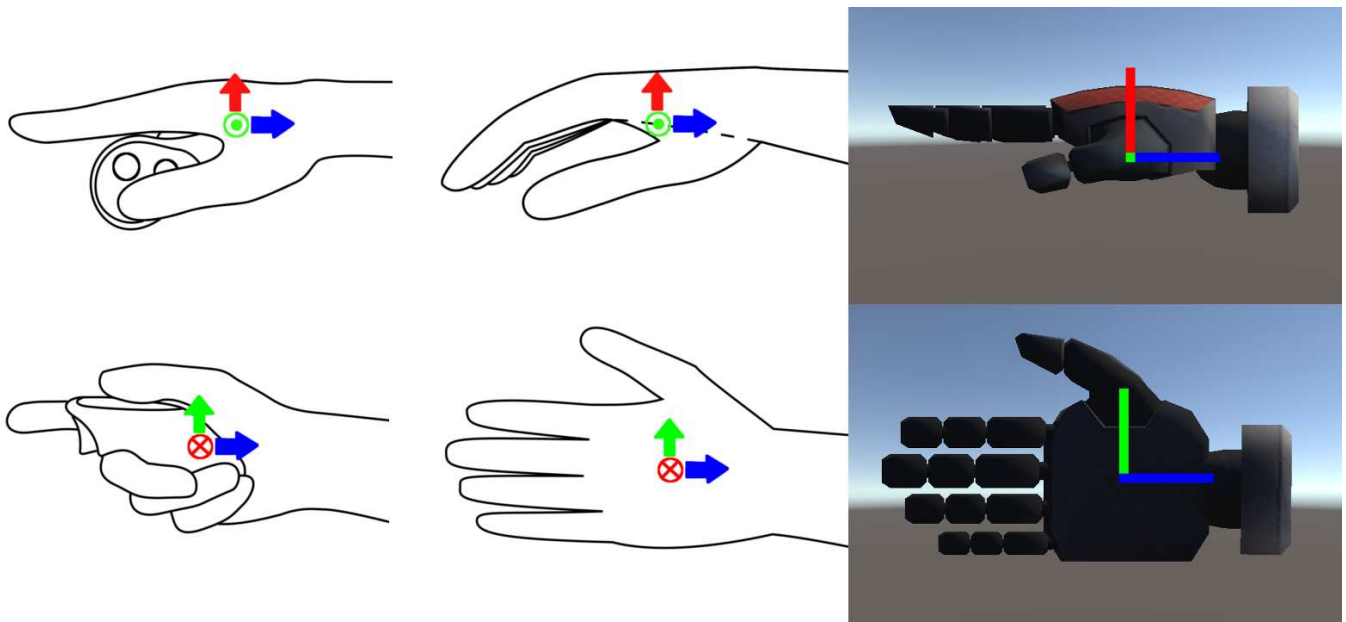


Figure 21. Example palm pose for (from left to right) a generic motion controller, tracked hand, and a digital hand avatar). The X axis is depicted in red. The Y axis is depicted in green. The Z axis is depicted in blue.

This pose is explicitly static for rigid controller type devices. The pose of `.../input/palm_ext/pose` and `.../input/grip_surface/pose` **must** be identical.

### Interaction Profile Additions

When this extension is active, a runtime **must** accept the `.../input/palm_ext/pose` component path for all interaction profiles that are valid for at least one of the user paths listed below listed below, including those interaction profiles enabled through extensions. Actions bound to such palm input component paths **must** behave as though those paths were listed in the original definition of an interaction profile.

Valid for the user paths

- `/user/hand/left`
- `/user/hand/right`

Supported component paths:

- On both user paths
  - `.../input/palm_ext/pose`

### Note

While this extension itself does not add the `.../input/palm_ext/pose` input component path to interaction profiles defined in extensions, extension authors **may** update existing extensions to add this path, or submit new extensions defining new interaction profiles using this pose identifier and component path. For consistency, it is recommended that the `.../input/palm_ext/pose` path in extension-defined interaction profiles be specified as only valid when this `XR_EXT_palm_pose` extension is also enabled.



This extension does pose a challenge to API layer implementers attempting to provide interaction profile support through their layer. If a runtime implements `XR_EXT_palm_pose`, and an application enables it, but such an API layer is unaware of it, the runtime may "accept" (not error) the additional suggested binding but the layer will not know to provide data or indicate an active binding. This behavior, while unexpected, does not violate the specification, and does not substantially increase the difficulty of providing additional input support using an API layer.

## Version History

- Revision 1, 2020-07-26 (Jack Pritz)
  - Initial extension proposal
- Revision 2, 2022-05-18 (Lachlan Ford)
  - Modification and cleanup of extension proposal based on working group discussion.
- Revision 3, 2023-11-16 (Ron Bessems)
  - Notes and clarification for the addition of `.../input/grip_surface/pose` to the core spec in OpenXR 1.1.

## 14.6. XR\_EXT\_samsung\_odyssey\_controller

### Name String

`XR_EXT_samsung_odyssey_controller`

### Extension Type

Instance extension

### Registered Extension Number

95

### Revision

1

## Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Deprecation State

- *Promoted* to [OpenXR 1.1](#)

### Last Modified Date

2020-06-08

### IP Status

No known IP claims.

### Contributors

Lachlan Ford, Microsoft

Alex Turner, Microsoft

Yin Li, Microsoft

Philippe Harscoet, Samsung Electronics

### Overview

This extension enables the application to differentiate the newer form factor of motion controller released with the Samsung Odyssey headset. It enables the application to customize the appearance and experience of the controller differently from the original [mixed reality motion controller](#).

This extension added a new interaction profile `/interaction_profiles/samsung/odyssey_controller` to describe the Odyssey controller. The action bindings of this interaction profile work exactly the same as the `/interaction_profiles/microsoft/motion_controller` in terms of valid user paths and supported input and output component paths.

#### Note

The interaction profile path `/interaction_profiles/samsung/odyssey_controller` defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called `/interaction_profiles/samsung/odyssey_controller_samsung`, to allow for modifications when promoted to a KHR extension or the core specification.

If the application does not do its own custom rendering for specific motion controllers, it **should** avoid using this extension and instead just use `.../microsoft/motion_controller`, as runtimes **should** treat both controllers equally when applications declare action bindings only for that profile.

If the application wants to customize rendering for specific motion controllers, it **should** setup the suggested bindings for `.../samsung/odyssey_controller` the same as `.../microsoft/motion_controller` when calling `xrSuggestInteractionProfileBindings`, and expect the same action bindings. Then the application **can** listen to the `XrEventDataInteractionProfileChanged` event and inspect the returned interaction profile from `xrGetCurrentInteractionProfile` to differentiate which controller is being used by the user, and hence customize the appearance or experience of the motion controller specifically for the form factor of `.../samsung/odyssey_controller`.

### Version History

- Revision 1, 2020-06-08 (Yin Li)
  - Initial extension proposal

## 14.7. XR\_EXT\_uuid

### Name String

`XR_EXT_uuid`

### Extension Type

Instance extension

### Registered Extension Number

300

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Deprecation State

- *Promoted* to [OpenXR 1.1](#)

### Last Modified Date

2021-10-27

### IP Status

No known IP claims.

### Contributors

Darryl Gough, Microsoft  
Yin Li, Microsoft  
Alex Turner, Microsoft  
David Fields, Microsoft

## Overview

This extension defines a Universally Unique Identifier that follows [RFC 4122](#).

The [XrUuidEXT](#) structure is a 128-bit Universally Unique Identifier and is defined as:

```
// Provided by XR_EXT_uuid
// XrUuidEXT is an alias for XrUuid
typedef struct XrUuid {
    uint8_t    data[XR_UUID_SIZE];
} XrUuid;

typedef XrUuid XrUuidEXT;
```

### Member Descriptions

- `data` is a 128-bit Universally Unique Identifier.

The structure is composed of 16 octets, with the size and order of the fields defined in [RFC 4122 section 4.1.2](#).

### Valid Usage (Implicit)

- The [XR\\_FB\\_spatial\\_entity](#) extension **must** be enabled prior to using [XrUuidEXT](#)

## New Object Types

## New Flag Types

## New Enum Constants

- `XR_UUID_SIZE_EXT`

## New Enums

## New Structures

- [XrUuidEXT](#)

## New Functions

## Issues

## Version History

- Revision 1, 2021-10-27 (Darryl Gough)
  - Initial extension description

# 14.8. XR\_BD\_controller\_interaction

## Name String

`XR_BD_controller_interaction`

## Extension Type

Instance extension

## Registered Extension Number

385

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

## Deprecation State

- *Promoted to* [OpenXR 1.1](#)

## Last Modified Date

2023-08-10

## IP Status\*

No known IP claims.

## Contributors

Baolin Fu, Bytedance  
Shanliang Xu, Bytedance  
Zhanrui Jia, Bytedance

## Overview

This extension defines the interaction profile for PICO Neo3, PICO 4, and PICO G3 Controllers.



## BD(ByteDance) Controller interaction profile

Interaction profile path for PICO Neo3:

- */interaction\_profiles/bytedance/pico\_neo3\_controller*

### Note

The interaction profile path */interaction\_profiles/bytedance/pico\_neo3\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/bytedance/pico\_neo3\_controller\_bd*, to allow for modifications when promoted to a KHR extension or the core specification.

Interaction profile path for PICO 4:

- */interaction\_profiles/bytedance/pico4\_controller*

### Note

The interaction profile path */interaction\_profiles/bytedance/pico4\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/bytedance/pico4\_controller\_bd*, to allow for modifications when promoted to a KHR extension or the core specification.

Interaction profile path for PICO G3:

- */interaction\_profiles/bytedance/pico\_g3\_controller*

### Note

The interaction profile path */interaction\_profiles/bytedance/pico\_g3\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/bytedance/pico\_g3\_controller\_bd*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths for *pico\_neo3\_controller*, *pico4\_controller*, and *pico\_g3\_controller*:

- */user/hand/left*
- */user/hand/right*

Supported component paths for *pico\_neo3\_controller*:

- On */user/hand/left* only:

- *.../input/x/click*
- *.../input/x/touch*
- *.../input/y/click*
- *.../input/y/touch*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/a/touch*
  - *.../input/b/click*
  - *.../input/b/touch*
- *.../input/menu/click*
- *.../input/system/click* (**may** not be available for application use)
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/thumbstick/y*
- *.../input/thumbstick/x*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/squeeze/click*
- *.../input/squeeze/value*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

Supported component paths for *pico4\_controller*:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/x/touch*
  - *.../input/y/click*
  - *.../input/y/touch*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*

- *.../input/a/touch*
- *.../input/b/click*
- *.../input/b/touch*
- *.../input/system/click* (**may** not be available for application use)
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/trigger/touch*
- *.../input/thumbstick/y*
- *.../input/thumbstick/x*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/squeeze/click*
- *.../input/squeeze/value*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*

Supported component paths for `pico_g3_controller`:

- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/menu/click*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../input/thumbstick*
- *.../input/thumbstick/click*

Be careful with the following difference:

- `pico_neo3_controller` supports *.../input/menu/click* both on `/user/hand/left` and `/user/hand/right`.
- `pico4_controller` supports *.../input/menu/click* only on `/user/hand/left`.
- `pico_g3_controller` has only one physical controller. When designing suggested bindings for this interaction profile, you **may** suggest bindings for both `/user/hand/left` and `/user/hand/right`. However, only one of them will be active at a given time, so do not design interactions that require simultaneous use of both hands.



#### Note

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`



#### Note

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`



#### Note

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



#### Note

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2023-01-04 (Baolin Fu)
  - Initial extension description
- Revision 2, 2023-08-10 (Shanliang Xu)

- Add support for G3 devices

## 14.9. XR\_HTC\_vive\_cosmos\_controller\_interaction

### Name String

`XR_HTC_vive_cosmos_controller_interaction`

### Extension Type

Instance extension

### Registered Extension Number

103

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Deprecation State

- *Promoted to [OpenXR 1.1](#)*

### Last Modified Date

2020-09-28

### IP Status

No known IP claims.

### Contributors

Chris Kuo, HTC

Kyle Chen, HTC

### Overview

This extension defines a new interaction profile for the VIVE Cosmos Controller.

### VIVE Cosmos Controller interaction profile

Interaction profile path:

- */interaction\_profiles/htc/vive\_cosmos\_controller*

### Note

The interaction profile path */interaction\_profiles/htc/vive\_cosmos\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/htc/vive\_cosmos\_controller\_htc*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the VIVE Cosmos Controller.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/y/click*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/b/click*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/shoulder/click*
- *.../input/squeeze/click*
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../output/haptic*



*Note*

When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- `.../input/grip_surface/pose`



*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2020-09-28 (Chris Kuo)
  - Initial extension description

## 14.10. XR\_HTC\_vive\_focus3\_controller\_interaction

### Name String

`XR_HTC_vive_focus3_controller_interaction`

### Extension Type

Instance extension

### Registered Extension Number

106

### Revision

2

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Deprecation State

- *Promoted to* [OpenXR 1.1](#)

### Last Modified Date

2022-04-29

### IP Status

No known IP claims.

### Contributors

Ria Hsu, HTC

### Overview

This extension defines a new interaction profile for the VIVE Focus 3 Controller.

### VIVE Focus 3 Controller interaction profile

Interaction profile path:

- `/interaction_profiles/htc/vive_focus3_controller`



## Note

The interaction profile path */interaction\_profiles/htc/vive\_focus3\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/htc/vive\_focus3\_controller\_htc*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

This interaction profile represents the input sources and haptics on the VIVE Focus 3 Controller.

Supported component paths:

- On */user/hand/left* only:
  - *.../input/x/click*
  - *.../input/y/click*
  - *.../input/menu/click*
- On */user/hand/right* only:
  - *.../input/a/click*
  - *.../input/b/click*
  - *.../input/system/click* (**may** not be available for application use)
- *.../input/squeeze/click*
- *.../input/squeeze/touch*
- *.../input/squeeze/value*
- *.../input/trigger/click*
- *.../input/trigger/touch*
- *.../input/trigger/value*
- *.../input/thumbstick/x*
- *.../input/thumbstick/y*
- *.../input/thumbstick/click*
- *.../input/thumbstick/touch*
- *.../input/thumbrest/touch*
- *.../input/grip/pose*

- *.../input/aim/pose*
- *.../output/haptic*

*Note*



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/grip\_surface/pose*

*Note*



When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/palm\_ext/pose*

*Note*



When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- *.../input/pinch\_ext/pose*
- *.../input/poke\_ext/pose*

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2022-01-03 (Ria Hsu)

- Initial extension description
- Revision 2, 2022-04-29 (Ria Hsu)
  - Support component path "/input/squeeze/value"

## 14.11. XR\_ML\_ml2\_controller\_interaction

### Name String

[XR\\_ML\\_ml2\\_controller\\_interaction](#)

### Extension Type

Instance extension

### Registered Extension Number

135

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### API Interactions

- Interacts with [XR\\_EXT\\_dpad\\_binding](#)
- Interacts with [XR\\_EXT\\_hand\\_interaction](#)
- Interacts with [XR\\_EXT\\_palm\\_pose](#)

### Deprecation State

- *Promoted* to [OpenXR 1.1](#)

### Last Modified Date

2022-07-22

### IP Status

No known IP claims.

### Contributors

Ron Bessems, Magic Leap  
Rafael Wiltz, Magic Leap

### Overview

This extension defines the interaction profile for the Magic Leap 2 Controller.

### Magic Leap 2 Controller interaction profile

This interaction profile represents the input sources and haptics on the Magic Leap 2 Controller.

Interaction profile path:

- */interaction\_profiles/ml/ml2\_controller*

### Note

The interaction profile path */interaction\_profiles/ml/ml2\_controller* defined here does not follow current rules for interaction profile names. If this extension were introduced today, it would be called */interaction\_profiles/ml/ml2\_controller\_ml*, to allow for modifications when promoted to a KHR extension or the core specification.

Valid for user paths:

- */user/hand/left*
- */user/hand/right*

Supported component paths:

- *.../input/menu/click*
- *.../input/home/click* (**may** not be available for application use)
- *.../input/trigger/click*
- *.../input/trigger/value*
- *.../input/trackpad/y*
- *.../input/trackpad/x*
- *.../input/trackpad/click*
- *.../input/trackpad/force*
- *.../input/trackpad/touch*
- *.../input/grip/pose*
- *.../input/aim/pose*
- *.../input/shoulder/click*
- *.../output/haptic*

#### Note



When the runtime supports [XR\\_VERSION\\_1\\_1](#) and use of OpenXR 1.1 is requested by the application, this interaction profile **must** also support

- *.../input/grip\_surface/pose*



*Note*

When the [XR\\_KHR\\_maintenance1](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/grip_surface/pose`



*Note*

When the [XR\\_EXT\\_palm\\_pose](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/palm_ext/pose`



*Note*

When the [XR\\_EXT\\_hand\\_interaction](#) extension is available and enabled, this interaction profile **must** also support

- `.../input/pinch_ext/pose`
- `.../input/poke_ext/pose`

## New Object Types

## New Flag Types

## New Enum Constants

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2022-07-22 (Ron Bessems)
  - Initial extension description

# 14.12. XR\_MND\_swapchain\_usage\_input\_attachment\_bit

## Name String

`XR_MND_swapchain_usage_input_attachment_bit`

## Extension Type

Instance extension

## Registered Extension Number

97

## Revision

2

## Extension and Version Dependencies

[OpenXR 1.0](#)

## Deprecation State

- *Deprecated* by [XR\\_KHR\\_swapchain\\_usage\\_input\\_attachment\\_bit](#) extension

## Last Modified Date

2020-07-24

## IP Status

No known IP claims.

## Contributors

Jakob Bornecrantz, Collabora

## Overview

This extension enables an application to specify that swapchain images should be created in a way so that they can be used as input attachments. At the time of writing this bit only affects Vulkan swapchains.

## New Object Types

## New Flag Types

## New Enum Constants

[XrSwapchainUsageFlagBits](#) enumeration is extended with:

- [XR\\_SWAPCHAIN\\_USAGE\\_INPUT\\_ATTACHMENT\\_BIT\\_MND](#)

## New Enums

## New Structures

## New Functions

## Issues

## Version History

- Revision 1, 2020-07-23 (Jakob Bornecrantz)
  - Initial draft
- Revision 2, 2020-07-24 (Jakob Bornecrantz)
  - Added note about only affecting Vulkan
  - Changed from MNDX to MND

## 14.13. XR\_OCULUS\_android\_session\_state\_enable

### Name String

`XR_OCULUS_android_session_state_enable`

### Extension Type

Instance extension

### Registered Extension Number

45

### Revision

1

### Extension and Version Dependencies

[OpenXR 1.0](#)

### Deprecation State

- *Deprecated* without replacement

### Overview

This extension enables the integration of the Android session lifecycle and an OpenXR runtime session state. Some OpenXR runtimes may require this extension to transition the application to the session READY or STOPPING state.

Applications that run on an Android system with this extension enabled have a different OpenXR Session state flow.

On Android, it is the Android Activity lifecycle that will dictate when the system is ready for the application to begin or end its session, not the runtime.

When XR\_OCULUS\_android\_session\_state is enabled, the following changes are made to Session State handling:

- The runtime does not determine when the application's session should be moved to the ready state,

`XR_SESSION_STATE_READY`. The application should not wait to receive the `XR_SESSION_STATE_READY` session state changed event before beginning a session. Instead, the application should begin their session once there is a surface and the activity is resumed.

- The application should not call `xrRequestExitSession` to request the session move to the stopping state, `XR_SESSION_STATE_STOPPING`. `xrRequestExitSession` will return `XR_ERROR_VALIDATION_FAILURE` if called.
- The application should not wait to receive the `XR_SESSION_STATE_STOPPING` session state changed event before ending a session. Instead, the application should end its session once the surface is destroyed or the activity is paused.
- The runtime will not transition to `XR_SESSION_STATE_READY` or `XR_SESSION_STATE_STOPPING` as the state is implicit from the Android activity and surface lifecycles.

## Android Activity life cycle

An Android Activity can only be in the session running state while the activity is in the resumed state. The following shows how beginning and ending an XR session fits into the Android Activity life cycle.

```
1. VrActivity::onCreate() <-----+
2. VrActivity::onStart() <-----+ |
3. VrActivity::onResume() <----+ | |
4. xrBeginSession()          | | |
5. xrEndSession()           | | |
6. VrActivity::onPause()  -----+ | |
7. VrActivity::onStop()  -----+ | |
8. VrActivity::onDestroy() -----+ |
```

## Android Surface life cycle

An Android Activity can only be in the session running state while there is a valid Android Surface. The following shows how beginning and ending an XR session fits into the Android Surface life cycle.

```
1. VrActivity::surfaceCreated() <----+
2. VrActivity::surfaceChanged()      |
3. xrBeginSession()                 |
4. xrEndSession()                   |
5. VrActivity::surfaceDestroyed() ---+
```

Note that the life cycle of a surface is not necessarily tightly coupled with the life cycle of an activity. These two life cycles may interleave in complex ways. Usually `surfaceCreated()` is called after `onResume()` and `surfaceDestroyed()` is called between `onPause()` and `onDestroy()`. However, this is not guaranteed and, for instance, `surfaceDestroyed()` may be called after `onDestroy()` or even before `onPause()`.



An Android Activity is only in the resumed state with a valid Android Surface between `surfaceChanged()` or `onResume()`, whichever comes last, and `surfaceDestroyed()` or `onPause()`, whichever comes first. In other words, a XR application will typically begin the session from `surfaceChanged()` or `onResume()`, whichever comes last, and end the session from `surfaceDestroyed()` or `onPause()`, whichever comes first.

## **New Object Types**

## **New Flag Types**

## **New Enum Constants**

## **New Enums**

## **New Structures**

## **New Functions**

## **Issues**

## **Version History**

- Revision 1, 2019-08-16 (Cass Everitt)
  - Initial extension description

# **14.14. XR\_VARJO\_quad\_views**

## **Name String**

`XR_VARJO_quad_views`

## **Extension Type**

Instance extension

## **Registered Extension Number**

38

## **Revision**

1

## **Extension and Version Dependencies**

[OpenXR 1.0](#)

## **Deprecation State**

- *Promoted to* [OpenXR 1.1](#)

## Last Modified Date

2019-04-16

## IP Status

No known IP claims.

## Contributors

Sergiy Dubovik, Varjo Technologies

Rémi Arnaud, Varjo Technologies

Robert Menzel, NVIDIA

### 14.14.1. Overview

This extension adds a new view configuration type - `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_QUAD_VARJO` to [`XrViewConfigurationType`](#) which can be returned by [`xrEnumerateViewConfigurations`](#) to indicate that the runtime supports 4 viewports.

In this configuration each eye consists of two viewports of which one is smaller (in terms of field of view) of the other and fully included inside of the larger FoV one. The small FoV viewport however can have a higher resolution with respect to the same field of view in the outer viewport. The motivation is special hardware which superimposes a smaller, high resolution screen for the fovea region onto a larger screen for the periphery.

The runtime guarantees that the inner viewport of each eye is fully inside of the outer viewport.

To enumerate the 4 views [`xrEnumerateViewConfigurationViews`](#) can be used. The first two views ([`XrViewConfigurationView`](#)) will be for the left and right eyes for the outer viewport. The views 2 and 3 are for the left and right eyes for the inner viewport.

The relative position of the inner views relative to the outer views can change at run-time.

The runtime might blend between the views at the edges, so the application should not omit the inner field of view from being generated in the outer view.

## New Object Types

## New Flag Types

## New Enum Constants

[`XrViewConfigurationType`](#) enumeration is extended with:

- `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_QUAD_VARJO`

## New Enums

## New Structures

## **New Functions**

## **Issues**

## **Version History**

- Revision 1, 2019-04-16 (Sergiy Dubovik)
  - Initial draft

# Chapter 15. Core Revisions (Informative)

New minor versions of the OpenXR API are defined periodically by the Khronos OpenXR Working Group. These consist of some amount of additional functionality added to the core API, potentially including both new functionality and functionality promoted from extensions.

## 15.1. Version 1.1

### 15.1.1. OpenXR 1.1 Promotions

OpenXR version 1.1 promoted a number of key extensions into the core API:

- [XR\\_KHR\\_locate\\_spaces](#)
- [XR\\_KHR\\_maintenance1](#)
- [XR\\_EXT\\_hp\\_mixed\\_reality\\_controller](#)
- [XR\\_EXT\\_local\\_floor](#)
- [XR\\_EXT\\_palm\\_pose](#)
- [XR\\_EXT\\_samsung\\_odyssey\\_controller](#)
- [XR\\_EXT\\_uuid](#)
- [XR\\_BD\\_controller\\_interaction](#)
- [XR\\_HTC\\_vive\\_cosmos\\_controller\\_interaction](#)
- [XR\\_HTC\\_vive\\_focus3\\_controller\\_interaction](#)
- [XR\\_ML\\_m12\\_controller\\_interaction](#)
- [XR\\_VARJO\\_quad\\_views](#)

All differences in behavior between these extensions and the corresponding OpenXR 1.1 functionality are summarized below.

#### Differences Relative to [XR\\_EXT\\_local\\_floor](#)

The definition of this space was made more precise, and it was clarified that the mandatory support of this space does **not** dictate any particular quality of floor level estimation. Applications that can provide a head-relative interaction experience in the absence of a defined stage continue to use **LOCAL** space, while those that need higher quality assertions about floor level continue to use **STAGE** space or scene understanding extensions to detect floor level. The (mandatory) presence of this space when enumerating reference spaces is a convenience for portability rather than an assertion that e.g. floor detection scene understanding has taken place or that the floor is inherently walkable.

## Differences Relative to [XR\\_EXT\\_palm\\_pose](#)

The input identifier `palm_ext` defined in the extension has been renamed to `grip_surface` to more clearly describe its intended use and distinguish it from hand tracking.

## Differences Relative to [XR\\_VARJO\\_quad\\_views](#)

The view configuration type enumerant `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_QUAD_VARJO` was renamed to `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET`, to clarify that it is not vendor-specific nor the only way four views are possible. In OpenXR 1.1, a runtime **may** support `XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO_WITH_FOVEATED_INSET`, but this is **optional** like the other view configuration types. Use [xrEnumerateViewConfigurations](#) to determine if it is provided, rather than using the presence or absence of the extension.

### 15.1.2. Additional OpenXR 1.1 Changes

In addition to the promoted extensions described above, OpenXR 1.1 changed the following:

- Substantial clarifications in the input and fundamentals chapters, intended to be non-substantive.
- Added the following legacy interaction profiles to represent specific controllers shipped under the Oculus/Meta Touch name and previously grouped into a single [Oculus Touch interaction profile](#):
  - `/interaction_profiles/meta/touch_controller_rift_cv1` - [Meta Touch Controller \(Rift CV1\) Profile](#)
  - `/interaction_profiles/meta/touch_controller_quest_1_rift_s` - [Meta Touch Controller \(Rift S / Quest 1\) Profile](#)
  - `/interaction_profiles/meta/touch_controller_quest_2` - [Meta Touch Controller \(Quest 2\) Profile](#)

### 15.1.3. New Commands

- [xrLocateSpaces](#)

### 15.1.4. New Structures

- [XrBoxf](#)
- [XrColor3f](#)
- [XrExtent3Df](#)
- [XrFrustumf](#)
- [XrSpaceLocationData](#)
- [XrSpaceLocations](#)
- [XrSpaceVelocityData](#)
- [XrSpacesLocateInfo](#)
- [XrSpheref](#)

- [XrUuid](#)
- Extending [XrSpaceLocations](#):
  - [XrSpaceVelocities](#)

### 15.1.5. New Enum Constants

- [XR\\_UUID\\_SIZE](#)
- Extending [XrReferenceSpaceType](#):
  - [XR\\_REFERENCE\\_SPACE\\_TYPE\\_LOCAL\\_FLOOR](#)
- Extending [XrResult](#):
  - [XR\\_ERROR\\_EXTENSION\\_DEPENDENCY\\_NOT\\_ENABLED](#)
  - [XR\\_ERROR\\_PERMISSION\\_INSUFFICIENT](#)
- Extending [XrStructureType](#):
  - [XR\\_TYPE\\_SPACES\\_LOCATE\\_INFO](#)
  - [XR\\_TYPE\\_SPACE\\_LOCATIONS](#)
  - [XR\\_TYPE\\_SPACE\\_VELOCITIES](#)
- Extending [XrViewConfigurationType](#):
  - [XR\\_VIEW\\_CONFIGURATION\\_TYPE\\_PRIMARY\\_STEREO\\_WITH\\_FOVEATED\\_INSET](#)

## 15.2. Loader Runtime and API Layer Negotiation Version 1.0

The OpenXR version 1.0.33 patch release included ratification of the runtime and API layer negotiation API, associated with the identifier [XR\\_LOADER\\_VERSION\\_1\\_0](#), substantially unchanged from the unratified form previously described in the loader design document. This interface is intended for use only between the loader, runtimes, and API layers, and is not typically directly used by an application.

### 15.2.1. New Macros

- [XR\\_API\\_LAYER\\_CREATE\\_INFO\\_STRUCT\\_VERSION](#)
- [XR\\_API\\_LAYER\\_INFO\\_STRUCT\\_VERSION](#)
- [XR\\_API\\_LAYER\\_MAX\\_SETTINGS\\_PATH\\_SIZE](#)
- [XR\\_API\\_LAYER\\_NEXT\\_INFO\\_STRUCT\\_VERSION](#)
- [XR\\_CURRENT\\_LOADER\\_API\\_LAYER\\_VERSION](#)
- [XR\\_CURRENT\\_LOADER\\_RUNTIME\\_VERSION](#)
- [XR\\_LOADER\\_INFO\\_STRUCT\\_VERSION](#)

- [XR\\_RUNTIME\\_INFO\\_STRUCT\\_VERSION](#)

### 15.2.2. New Commands

- [xrCreateApiLayerInstance](#)
- [xrNegotiateLoaderApiLayerInterface](#)
- [xrNegotiateLoaderRuntimeInterface](#)

## 15.3. Version 1.0

OpenXR version 1.0 defined the initial core API.

### 15.3.1. New Macros

- [XR\\_CURRENT\\_API\\_VERSION](#)
- [XR\\_DEFINE\\_HANDLE](#)
- [XR\\_DEFINE\\_OPAQUE\\_64](#)
- [XR\\_EXTENSION\\_ENUM\\_BASE](#)
- [XR\\_EXTENSION\\_ENUM\\_STRIDE](#)
- [XR\\_FAILED](#)
- [XR\\_FREQUENCY\\_UNSPECIFIED](#)
- [XR\\_INFINITE\\_DURATION](#)
- [XR\\_MAX\\_EVENT\\_DATA\\_SIZE](#)
- [XR\\_MAY\\_ALIAS](#)
- [XR\\_MIN\\_COMPOSITION\\_LAYERS\\_SUPPORTED](#)
- [XR\\_MIN\\_HAPTIC\\_DURATION](#)
- [XR\\_NO\\_DURATION](#)
- [XR\\_NULL\\_HANDLE](#)
- [XR\\_NULL\\_PATH](#)
- [XR\\_NULL\\_SYSTEM\\_ID](#)
- [XR\\_SUCCEEDED](#)
- [XR\\_UNQUALIFIED\\_SUCCESS](#)
- [XR\\_VERSION\\_MAJOR](#)
- [XR\\_VERSION\\_MINOR](#)
- [XR\\_VERSION\\_PATCH](#)

## 15.3.2. New Base Types

- [XrVersion](#)

## 15.3.3. New Commands

- [xrAcquireSwapchainImage](#)
- [xrApplyHapticFeedback](#)
- [xrAttachSessionActionSets](#)
- [xrBeginFrame](#)
- [xrBeginSession](#)
- [xrCreateAction](#)
- [xrCreateActionSet](#)
- [xrCreateActionSpace](#)
- [xrCreateInstance](#)
- [xrCreateReferenceSpace](#)
- [xrCreateSession](#)
- [xrCreateSwapchain](#)
- [xrDestroyAction](#)
- [xrDestroyActionSet](#)
- [xrDestroyInstance](#)
- [xrDestroySession](#)
- [xrDestroySpace](#)
- [xrDestroySwapchain](#)
- [xrEndFrame](#)
- [xrEndSession](#)
- [xrEnumerateApiLayerProperties](#)
- [xrEnumerateBoundSourcesForAction](#)
- [xrEnumerateEnvironmentBlendModes](#)
- [xrEnumerateInstanceExtensionProperties](#)
- [xrEnumerateReferenceSpaces](#)
- [xrEnumerateSwapchainFormats](#)
- [xrEnumerateSwapchainImages](#)
- [xrEnumerateViewConfigurationViews](#)



- [xrEnumerateViewConfigurations](#)
- [xrGetActionStateBoolean](#)
- [xrGetActionStateFloat](#)
- [xrGetActionStatePose](#)
- [xrGetActionStateVector2f](#)
- [xrGetCurrentInteractionProfile](#)
- [xrGetInputSourceLocalizedName](#)
- [xrGetInstanceProcAddr](#)
- [xrGetInstanceProperties](#)
- [xrGetReferenceSpaceBoundsRect](#)
- [xrGetSystem](#)
- [xrGetSystemProperties](#)
- [xrGetViewConfigurationProperties](#)
- [xrLocateSpace](#)
- [xrLocateViews](#)
- [xrPathToString](#)
- [xrPollEvent](#)
- [xrReleaseSwapchainImage](#)
- [xrRequestExitSession](#)
- [xrResultToString](#)
- [xrStopHapticFeedback](#)
- [xrStringToPath](#)
- [xrStructureTypeToString](#)
- [xrSuggestInteractionProfileBindings](#)
- [xrSyncActions](#)
- [xrWaitFrame](#)
- [xrWaitSwapchainImage](#)

#### 15.3.4. New Structures

- [XrBaseInStructure](#)
- [XrBaseOutStructure](#)
- [XrColor4f](#)
- [XrCompositionLayerProjection](#)

- [XrCompositionLayerQuad](#)
- [XrEventDataBaseHeader](#)
- [XrEventDataEventsLost](#)
- [XrEventDataInstanceLossPending](#)
- [XrEventDataInteractionProfileChanged](#)
- [XrEventDataReferenceSpaceChangePending](#)
- [XrEventDataSessionStateChanged](#)
- [XrExtent2Df](#)
- [XrHapticVibration](#)
- [XrOffset2Df](#)
- [XrRect2Df](#)
- [XrVector4f](#)
- Extending [XrSpaceLocation](#):
  - [XrSpaceVelocity](#)

### 15.3.5. New Enums

- [XrObjectType](#)

### 15.3.6. New Headers

- [openxr\\_platform\\_defines](#)

### 15.3.7. New Enum Constants

- [XR\\_FALSE](#)
- [XR\\_MAX\\_API\\_LAYER\\_DESCRIPTION\\_SIZE](#)
- [XR\\_MAX\\_API\\_LAYER\\_NAME\\_SIZE](#)
- [XR\\_MAX\\_APPLICATION\\_NAME\\_SIZE](#)
- [XR\\_MAX\\_ENGINE\\_NAME\\_SIZE](#)
- [XR\\_MAX\\_EXTENSION\\_NAME\\_SIZE](#)
- [XR\\_MAX\\_PATH\\_LENGTH](#)
- [XR\\_MAX\\_RESULT\\_STRING\\_SIZE](#)
- [XR\\_MAX\\_RUNTIME\\_NAME\\_SIZE](#)
- [XR\\_MAX\\_STRUCTURE\\_NAME\\_SIZE](#)
- [XR\\_MAX\\_SYSTEM\\_NAME\\_SIZE](#)

- XR\_TRUE

# Appendix

## Code Style Conventions

These are the code style conventions used in this specification to define the API.

### Conventions

- Enumerants and defines are all upper case with words separated by an underscore.
- Neither type, function or member names contain underscores.
- Structure members start with a lower case character and each consecutive word starts with a capital.
- A structure that has a pointer to an array includes a structure member named `fooCount` of type `uint32_t` to denote the number of elements in the array of `foo`.
- A structure that has a pointer to an array lists the `fooCount` member first and then the array pointer.
- Unless a negative value has a clearly defined meaning all `fooCount` variables are unsigned.
- Function parameters that are modified are always listed last.

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

| Prefix              | Description  |
|---------------------|--|
| <code>XR_</code>    | Enumerants and defines are prefixed with these characters.     |
| <code>Xr</code>     | Non-function-pointer types are prefixed with these characters. |
| <code>xr</code>     | Functions are prefixed with these characters.                  |
| <code>PFN_xr</code> | Function pointer types are prefixed with these characters.     |

## Application Binary Interface

This section describes additional definitions and conventions that define the application binary interface.

## Structure Types

```
typedef enum XrStructureType {
    XR_TYPE_UNKNOWN = 0,
    XR_TYPE_API_LAYER_PROPERTIES = 1,
    XR_TYPE_EXTENSION_PROPERTIES = 2,
    XR_TYPE_INSTANCE_CREATE_INFO = 3,
    XR_TYPE_SYSTEM_GET_INFO = 4,
    XR_TYPE_SYSTEM_PROPERTIES = 5,
    XR_TYPE_VIEW_LOCATE_INFO = 6,
    XR_TYPE_VIEW = 7,
    XR_TYPE_SESSION_CREATE_INFO = 8,
    XR_TYPE_SWAPCHAIN_CREATE_INFO = 9,
    XR_TYPE_SESSION_BEGIN_INFO = 10,
    XR_TYPE_VIEW_STATE = 11,
    XR_TYPE_FRAME_END_INFO = 12,
    XR_TYPE_HAPTIC_VIBRATION = 13,
    XR_TYPE_EVENT_DATA_BUFFER = 16,
    XR_TYPE_EVENT_DATA_INSTANCE_LOSS_PENDING = 17,
    XR_TYPE_EVENT_DATA_SESSION_STATE_CHANGED = 18,
    XR_TYPE_ACTION_STATE_BOOLEAN = 23,
    XR_TYPE_ACTION_STATE_FLOAT = 24,
    XR_TYPE_ACTION_STATE_VECTOR2F = 25,
    XR_TYPE_ACTION_STATE_POSE = 27,
    XR_TYPE_ACTION_SET_CREATE_INFO = 28,
    XR_TYPE_ACTION_CREATE_INFO = 29,
    XR_TYPE_INSTANCE_PROPERTIES = 32,
    XR_TYPE_FRAME_WAIT_INFO = 33,
    XR_TYPE_COMPOSITION_LAYER_PROJECTION = 35,
    XR_TYPE_COMPOSITION_LAYER_QUAD = 36,
    XR_TYPE_REFERENCE_SPACE_CREATE_INFO = 37,
    XR_TYPE_ACTION_SPACE_CREATE_INFO = 38,
    XR_TYPE_EVENT_DATA_REFERENCE_SPACE_CHANGE_PENDING = 40,
    XR_TYPE_VIEW_CONFIGURATION_VIEW = 41,
    XR_TYPE_SPACE_LOCATION = 42,
    XR_TYPE_SPACE_VELOCITY = 43,
    XR_TYPE_FRAME_STATE = 44,
    XR_TYPE_VIEW_CONFIGURATION_PROPERTIES = 45,
    XR_TYPE_FRAME_BEGIN_INFO = 46,
    XR_TYPE_COMPOSITION_LAYER_PROJECTION_VIEW = 48,
    XR_TYPE_EVENT_DATA_EVENTS_LOST = 49,
    XR_TYPE_INTERACTION_PROFILE_SUGGESTED_BINDING = 51,
    XR_TYPE_EVENT_DATA_INTERACTION_PROFILE_CHANGED = 52,
    XR_TYPE_INTERACTION_PROFILE_STATE = 53,
    XR_TYPE_SWAPCHAIN_IMAGE_ACQUIRE_INFO = 55,
```

```

XR_TYPE_SWAPCHAIN_IMAGE_WAIT_INFO = 56,
XR_TYPE_SWAPCHAIN_IMAGE_RELEASE_INFO = 57,
XR_TYPE_ACTION_STATE_GET_INFO = 58,
XR_TYPE_HAPTIC_ACTION_INFO = 59,
XR_TYPE_SESSION_ACTION_SETS_ATTACH_INFO = 60,
XR_TYPE_ACTIONS_SYNC_INFO = 61,
XR_TYPE_BOUND_SOURCES_FOR_ACTION_ENUMERATE_INFO = 62,
XR_TYPE_INPUT_SOURCE_LOCALIZED_NAME_GET_INFO = 63,
// Provided by XR_VERSION_1_1
XR_TYPE_SPACES_LOCATE_INFO = 1000471000,
// Provided by XR_VERSION_1_1
XR_TYPE_SPACE_LOCATIONS = 1000471001,
// Provided by XR_VERSION_1_1
XR_TYPE_SPACE_VELOCITIES = 1000471002,
// Provided by XR_KHR_composition_layer_cube
XR_TYPE_COMPOSITION_LAYER_CUBE_KHR = 1000006000,
// Provided by XR_KHR_android_create_instance
XR_TYPE_INSTANCE_CREATE_INFO_ANDROID_KHR = 1000008000,
// Provided by XR_KHR_composition_layer_depth
XR_TYPE_COMPOSITION_LAYER_DEPTH_INFO_KHR = 1000010000,
// Provided by XR_KHR_vulkan_swapchain_format_list
XR_TYPE_VULKAN_SWAPCHAIN_FORMAT_LIST_CREATE_INFO_KHR = 1000014000,
// Provided by XR_EXT_performance_settings
XR_TYPE_EVENT_DATA_PERF_SETTINGS_EXT = 1000015000,
// Provided by XR_KHR_composition_layer_cylinder
XR_TYPE_COMPOSITION_LAYER_CYLINDER_KHR = 1000017000,
// Provided by XR_KHR_composition_layer_equirect
XR_TYPE_COMPOSITION_LAYER_EQUIRECT_KHR = 1000018000,
// Provided by XR_EXT_debug_utils
XR_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT = 1000019000,
// Provided by XR_EXT_debug_utils
XR_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT = 1000019001,
// Provided by XR_EXT_debug_utils
XR_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT = 1000019002,
// Provided by XR_EXT_debug_utils
XR_TYPE_DEBUG_UTILS_LABEL_EXT = 1000019003,
// Provided by XR_KHR_opengl_enable
XR_TYPE_GRAPHICS_BINDING_OPENGL_WIN32_KHR = 1000023000,
// Provided by XR_KHR_opengl_enable
XR_TYPE_GRAPHICS_BINDING_OPENGL_XLIB_KHR = 1000023001,
// Provided by XR_KHR_opengl_enable
XR_TYPE_GRAPHICS_BINDING_OPENGL_XCB_KHR = 1000023002,
// Provided by XR_KHR_opengl_enable
XR_TYPE_GRAPHICS_BINDING_OPENGL_WAYLAND_KHR = 1000023003,
// Provided by XR_KHR_opengl_enable
XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_KHR = 1000023004,
// Provided by XR_KHR_opengl_enable
XR_TYPE_GRAPHICS_REQUIREMENTS_OPENGL_KHR = 1000023005,

```

```

// Provided by XR_KHR_opengl_es_enable
XR_TYPE_GRAPHICS_BINDING_OPENGL_ES_ANDROID_KHR = 1000024001,
// Provided by XR_KHR_opengl_es_enable
XR_TYPE_SWAPCHAIN_IMAGE_OPENGL_ES_KHR = 1000024002,
// Provided by XR_KHR_opengl_es_enable
XR_TYPE_GRAPHICS_REQUIREMENTS_OPENGL_ES_KHR = 1000024003,
// Provided by XR_KHR_vulkan_enable
XR_TYPE_GRAPHICS_BINDING_VULKAN_KHR = 1000025000,
// Provided by XR_KHR_vulkan_enable
XR_TYPE_SWAPCHAIN_IMAGE_VULKAN_KHR = 1000025001,
// Provided by XR_KHR_vulkan_enable
XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN_KHR = 1000025002,
// Provided by XR_KHR_D3D11_enable
XR_TYPE_GRAPHICS_BINDING_D3D11_KHR = 1000027000,
// Provided by XR_KHR_D3D11_enable
XR_TYPE_SWAPCHAIN_IMAGE_D3D11_KHR = 1000027001,
// Provided by XR_KHR_D3D11_enable
XR_TYPE_GRAPHICS_REQUIREMENTS_D3D11_KHR = 1000027002,
// Provided by XR_KHR_D3D12_enable
XR_TYPE_GRAPHICS_BINDING_D3D12_KHR = 1000028000,
// Provided by XR_KHR_D3D12_enable
XR_TYPE_SWAPCHAIN_IMAGE_D3D12_KHR = 1000028001,
// Provided by XR_KHR_D3D12_enable
XR_TYPE_GRAPHICS_REQUIREMENTS_D3D12_KHR = 1000028002,
// Provided by XR_EXT_eye_gaze_interaction
XR_TYPE_SYSTEM_EYE_GAZE_INTERACTION_PROPERTIES_EXT = 1000030000,
// Provided by XR_EXT_eye_gaze_interaction
XR_TYPE_EYE_GAZE_SAMPLE_TIME_EXT = 1000030001,
// Provided by XR_KHR_visibility_mask
XR_TYPE_VISIBILITY_MASK_KHR = 1000031000,
// Provided by XR_KHR_visibility_mask
XR_TYPE_EVENT_DATA_VISIBILITY_MASK_CHANGED_KHR = 1000031001,
// Provided by XR_EXTX_overlay
XR_TYPE_SESSION_CREATE_INFO_OVERLAY_EXTX = 1000033000,
// Provided by XR_EXTX_overlay
XR_TYPE_EVENT_DATA_MAIN_SESSION_VISIBILITY_CHANGED_EXTX = 1000033003,
// Provided by XR_KHR_composition_layer_color_scale_bias
XR_TYPE_COMPOSITION_LAYER_COLOR_SCALE_BIAS_KHR = 1000034000,
// Provided by XR_MSFT_spatial_anchor
XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_MSFT = 1000039000,
// Provided by XR_MSFT_spatial_anchor
XR_TYPE_SPATIAL_ANCHOR_SPACE_CREATE_INFO_MSFT = 1000039001,
// Provided by XR_FB_composition_layer_image_layout
XR_TYPE_COMPOSITION_LAYER_IMAGE_LAYOUT_FB = 1000040000,
// Provided by XR_FB_composition_layer_alpha_blend
XR_TYPE_COMPOSITION_LAYER_ALPHA_BLEND_FB = 1000041001,
// Provided by XR_EXT_view_configuration_depth_range
XR_TYPE_VIEW_CONFIGURATION_DEPTH_RANGE_EXT = 1000046000,

```

```

// Provided by XR_MNDX_egl_enable
XR_TYPE_GRAPHICS_BINDING_EGL_MNDX = 1000048004,
// Provided by XR_MSFT_spatial_graph_bridge
XR_TYPE_SPATIAL_GRAPH_NODE_SPACE_CREATE_INFO_MSFT = 1000049000,
// Provided by XR_MSFT_spatial_graph_bridge
XR_TYPE_SPATIAL_GRAPH_STATIC_NODE_BINDING_CREATE_INFO_MSFT = 1000049001,
// Provided by XR_MSFT_spatial_graph_bridge
XR_TYPE_SPATIAL_GRAPH_NODE_BINDING_PROPERTIES_GET_INFO_MSFT = 1000049002,
// Provided by XR_MSFT_spatial_graph_bridge
XR_TYPE_SPATIAL_GRAPH_NODE_BINDING_PROPERTIES_MSFT = 1000049003,
// Provided by XR_EXT_hand_tracking
XR_TYPE_SYSTEM_HAND_TRACKING_PROPERTIES_EXT = 1000051000,
// Provided by XR_EXT_hand_tracking
XR_TYPE_HAND_TRACKER_CREATE_INFO_EXT = 1000051001,
// Provided by XR_EXT_hand_tracking
XR_TYPE_HAND_JOINTS_LOCATE_INFO_EXT = 1000051002,
// Provided by XR_EXT_hand_tracking
XR_TYPE_HAND_JOINT_LOCATIONS_EXT = 1000051003,
// Provided by XR_EXT_hand_tracking
XR_TYPE_HAND_JOINT_VELOCITIES_EXT = 1000051004,
// Provided by XR_MSFT_hand_tracking_mesh
XR_TYPE_SYSTEM_HAND_TRACKING_MESH_PROPERTIES_MSFT = 1000052000,
// Provided by XR_MSFT_hand_tracking_mesh
XR_TYPE_HAND_MESH_SPACE_CREATE_INFO_MSFT = 1000052001,
// Provided by XR_MSFT_hand_tracking_mesh
XR_TYPE_HAND_MESH_UPDATE_INFO_MSFT = 1000052002,
// Provided by XR_MSFT_hand_tracking_mesh
XR_TYPE_HAND_MESH_MSFT = 1000052003,
// Provided by XR_MSFT_hand_tracking_mesh
XR_TYPE_HAND_POSE_TYPE_INFO_MSFT = 1000052004,
// Provided by XR_MSFT_secondary_view_configuration
XR_TYPE_SECONDARY_VIEW_CONFIGURATION_SESSION_BEGIN_INFO_MSFT = 1000053000,
// Provided by XR_MSFT_secondary_view_configuration
XR_TYPE_SECONDARY_VIEW_CONFIGURATION_STATE_MSFT = 1000053001,
// Provided by XR_MSFT_secondary_view_configuration
XR_TYPE_SECONDARY_VIEW_CONFIGURATION_FRAME_STATE_MSFT = 1000053002,
// Provided by XR_MSFT_secondary_view_configuration
XR_TYPE_SECONDARY_VIEW_CONFIGURATION_FRAME_END_INFO_MSFT = 1000053003,
// Provided by XR_MSFT_secondary_view_configuration
XR_TYPE_SECONDARY_VIEW_CONFIGURATION_LAYER_INFO_MSFT = 1000053004,
// Provided by XR_MSFT_secondary_view_configuration
XR_TYPE_SECONDARY_VIEW_CONFIGURATION_SWAPCHAIN_CREATE_INFO_MSFT = 1000053005,
// Provided by XR_MSFT_controller_model
XR_TYPE_CONTROLLER_MODEL_KEY_STATE_MSFT = 1000055000,
// Provided by XR_MSFT_controller_model
XR_TYPE_CONTROLLER_MODEL_NODE_PROPERTIES_MSFT = 1000055001,
// Provided by XR_MSFT_controller_model
XR_TYPE_CONTROLLER_MODEL_PROPERTIES_MSFT = 1000055002,

```



```

// Provided by XR_MSFT_controller_model
XR_TYPE_CONTROLLER_MODEL_NODE_STATE_MSFT = 1000055003,
// Provided by XR_MSFT_controller_model
XR_TYPE_CONTROLLER_MODEL_STATE_MSFT = 1000055004,
// Provided by XR_EPIC_view_configuration_fov
XR_TYPE_VIEW_CONFIGURATION_VIEW_FOV_EPIC = 1000059000,
// Provided by XR_MSFT_holographic_window_attachment
XR_TYPE_HOLOGRAPHIC_WINDOW_ATTACHMENT_MSFT = 1000063000,
// Provided by XR_MSFT_composition_layer_reprojection
XR_TYPE_COMPOSITION_LAYER_REPROJECTION_INFO_MSFT = 1000066000,
// Provided by XR_MSFT_composition_layer_reprojection
XR_TYPE_COMPOSITION_LAYER_REPROJECTION_PLANE_OVERRIDE_MSFT = 1000066001,
// Provided by XR_FB_android_surface_swapchain_create
XR_TYPE_ANDROID_SURFACE_SWAPCHAIN_CREATE_INFO_FB = 1000070000,
// Provided by XR_FB_composition_layer_secure_content
XR_TYPE_COMPOSITION_LAYER_SECURE_CONTENT_FB = 1000072000,
// Provided by XR_FB_body_tracking
XR_TYPE_BODY_TRACKER_CREATE_INFO_FB = 1000076001,
// Provided by XR_FB_body_tracking
XR_TYPE_BODY_JOINTS_LOCATE_INFO_FB = 1000076002,
// Provided by XR_FB_body_tracking
XR_TYPE_SYSTEM_BODY_TRACKING_PROPERTIES_FB = 1000076004,
// Provided by XR_FB_body_tracking
XR_TYPE_BODY_JOINT_LOCATIONS_FB = 1000076005,
// Provided by XR_FB_body_tracking
XR_TYPE_BODY_SKELETON_FB = 1000076006,
// Provided by XR_EXT_dpad_binding
XR_TYPE_INTERACTION_PROFILE_DPAD_BINDING_EXT = 1000078000,
// Provided by XR_VALVE_analog_threshold
XR_TYPE_INTERACTION_PROFILE_ANALOG_THRESHOLD_VALVE = 1000079000,
// Provided by XR_EXT_hand_joints_motion_range
XR_TYPE_HAND_JOINTS_MOTION_RANGE_INFO_EXT = 1000080000,
// Provided by XR_KHR_loader_init_android
XR_TYPE_LOADER_INIT_INFO_ANDROID_KHR = 1000089000,
// Provided by XR_KHR_vulkan_enable2
XR_TYPE_VULKAN_INSTANCE_CREATE_INFO_KHR = 1000090000,
// Provided by XR_KHR_vulkan_enable2
XR_TYPE_VULKAN_DEVICE_CREATE_INFO_KHR = 1000090001,
// Provided by XR_KHR_vulkan_enable2
XR_TYPE_VULKAN_GRAPHICS_DEVICE_GET_INFO_KHR = 1000090003,
// Provided by XR_KHR_composition_layer_equirect2
XR_TYPE_COMPOSITION_LAYER_EQUIRECT2_KHR = 1000091000,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_OBSERVER_CREATE_INFO_MSFT = 1000097000,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_CREATE_INFO_MSFT = 1000097001,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_NEW_SCENE_COMPUTE_INFO_MSFT = 1000097002,

```

```

// Provided by XR_MSFT_scene_understanding
XR_TYPE_VISUAL_MESH_COMPUTE_LOD_INFO_MSFT = 1000097003,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_COMPONENTS_MSFT = 1000097004,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_COMPONENTS_GET_INFO_MSFT = 1000097005,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_COMPONENT_LOCATIONS_MSFT = 1000097006,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_COMPONENTS_LOCATE_INFO_MSFT = 1000097007,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_OBJECTS_MSFT = 1000097008,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_COMPONENT_PARENT_FILTER_INFO_MSFT = 1000097009,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_OBJECT_TYPES_FILTER_INFO_MSFT = 1000097010,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_PLANES_MSFT = 1000097011,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_PLANE_ALIGNMENT_FILTER_INFO_MSFT = 1000097012,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_MESHES_MSFT = 1000097013,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_MESH_BUFFERS_GET_INFO_MSFT = 1000097014,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_MESH_BUFFERS_MSFT = 1000097015,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_MESH_VERTEX_BUFFER_MSFT = 1000097016,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_MESH_INDICES_UINT32_MSFT = 1000097017,
// Provided by XR_MSFT_scene_understanding
XR_TYPE_SCENE_MESH_INDICES_UINT16_MSFT = 1000097018,
// Provided by XR_MSFT_scene_understanding_serialization
XR_TYPE_SERIALIZED_SCENE_FRAGMENT_DATA_GET_INFO_MSFT = 1000098000,
// Provided by XR_MSFT_scene_understanding_serialization
XR_TYPE_SCENE_DESERIALIZE_INFO_MSFT = 1000098001,
// Provided by XR_FB_display_refresh_rate
XR_TYPE_EVENT_DATA_DISPLAY_REFRESH_RATE_CHANGED_FB = 1000101000,
// Provided by XR_HTCX_vive_tracker_interaction
XR_TYPE_VIVE_TRACKER_PATHS_HTCX = 1000103000,
// Provided by XR_HTCX_vive_tracker_interaction
XR_TYPE_EVENT_DATA_VIVE_TRACKER_CONNECTED_HTCX = 1000103001,
// Provided by XR_HTC_facial_tracking
XR_TYPE_SYSTEM_FACIAL_TRACKING_PROPERTIES_HTC = 1000104000,
// Provided by XR_HTC_facial_tracking
XR_TYPE_FACIAL_TRACKER_CREATE_INFO_HTC = 1000104001,
// Provided by XR_HTC_facial_tracking
XR_TYPE_FACIAL_EXPRESSIONS_HTC = 1000104002,

```

```

// Provided by XR_FB_color_space
XR_TYPE_SYSTEM_COLOR_SPACE_PROPERTIES_FB = 1000108000,
// Provided by XR_FB_hand_tracking_mesh
XR_TYPE_HAND_TRACKING_MESH_FB = 1000110001,
// Provided by XR_FB_hand_tracking_mesh
XR_TYPE_HAND_TRACKING_SCALE_FB = 1000110003,
// Provided by XR_FB_hand_tracking_aim
XR_TYPE_HAND_TRACKING_AIM_STATE_FB = 1000111001,
// Provided by XR_FB_hand_tracking_capsules
XR_TYPE_HAND_TRACKING_CAPSULES_STATE_FB = 1000112000,
// Provided by XR_FB_spatial_entity
XR_TYPE_SYSTEM_SPATIAL_ENTITY_PROPERTIES_FB = 1000113004,
// Provided by XR_FB_spatial_entity
XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_FB = 1000113003,
// Provided by XR_FB_spatial_entity
XR_TYPE_SPACE_COMPONENT_STATUS_SET_INFO_FB = 1000113007,
// Provided by XR_FB_spatial_entity
XR_TYPE_SPACE_COMPONENT_STATUS_FB = 1000113001,
// Provided by XR_FB_spatial_entity
XR_TYPE_EVENT_DATA_SPATIAL_ANCHOR_CREATE_COMPLETE_FB = 1000113005,
// Provided by XR_FB_spatial_entity
XR_TYPE_EVENT_DATA_SPACE_SET_STATUS_COMPLETE_FB = 1000113006,
// Provided by XR_FB_foveation
XR_TYPE_FOVEATION_PROFILE_CREATE_INFO_FB = 1000114000,
// Provided by XR_FB_foveation
XR_TYPE_SWAPCHAIN_CREATE_INFO_FOVEATION_FB = 1000114001,
// Provided by XR_FB_foveation
XR_TYPE_SWAPCHAIN_STATE_FOVEATION_FB = 1000114002,
// Provided by XR_FB_foveation_configuration
XR_TYPE_FOVEATION_LEVEL_PROFILE_CREATE_INFO_FB = 1000115000,
// Provided by XR_FB_keyboard_tracking
XR_TYPE_KEYBOARD_SPACE_CREATE_INFO_FB = 1000116009,
// Provided by XR_FB_keyboard_tracking
XR_TYPE_KEYBOARD_TRACKING_QUERY_FB = 1000116004,
// Provided by XR_FB_keyboard_tracking
XR_TYPE_SYSTEM_KEYBOARD_TRACKING_PROPERTIES_FB = 1000116002,
// Provided by XR_FB_triangle_mesh
XR_TYPE_TRIANGLE_MESH_CREATE_INFO_FB = 1000117001,
// Provided by XR_FB_passthrough
XR_TYPE_SYSTEM_PASSTHROUGH_PROPERTIES_FB = 1000118000,
// Provided by XR_FB_passthrough
XR_TYPE_PASSTHROUGH_CREATE_INFO_FB = 1000118001,
// Provided by XR_FB_passthrough
XR_TYPE_PASSTHROUGH_LAYER_CREATE_INFO_FB = 1000118002,
// Provided by XR_FB_passthrough
XR_TYPE_COMPOSITION_LAYER_PASSTHROUGH_FB = 1000118003,
// Provided by XR_FB_passthrough
XR_TYPE_GEOMETRY_INSTANCE_CREATE_INFO_FB = 1000118004,

```

```

// Provided by XR_FB_passthrough
XR_TYPE_GEOMETRY_INSTANCE_TRANSFORM_FB = 1000118005,
// Provided by XR_FB_passthrough
XR_TYPE_SYSTEM_PASSTHROUGH_PROPERTIES2_FB = 1000118006,
// Provided by XR_FB_passthrough
XR_TYPE_PASSTHROUGH_STYLE_FB = 1000118020,
// Provided by XR_FB_passthrough
XR_TYPE_PASSTHROUGH_COLOR_MAP_MONO_TO_RGBA_FB = 1000118021,
// Provided by XR_FB_passthrough
XR_TYPE_PASSTHROUGH_COLOR_MAP_MONO_TO_MONO_FB = 1000118022,
// Provided by XR_FB_passthrough
XR_TYPE_PASSTHROUGH_BRIGHTNESS_CONTRAST_SATURATION_FB = 1000118023,
// Provided by XR_FB_passthrough
XR_TYPE_EVENT_DATA_PASSTHROUGH_STATE_CHANGED_FB = 1000118030,
// Provided by XR_FB_render_model
XR_TYPE_RENDER_MODEL_PATH_INFO_FB = 1000119000,
// Provided by XR_FB_render_model
XR_TYPE_RENDER_MODEL_PROPERTIES_FB = 1000119001,
// Provided by XR_FB_render_model
XR_TYPE_RENDER_MODEL_BUFFER_FB = 1000119002,
// Provided by XR_FB_render_model
XR_TYPE_RENDER_MODEL_LOAD_INFO_FB = 1000119003,
// Provided by XR_FB_render_model
XR_TYPE_SYSTEM_RENDER_MODEL_PROPERTIES_FB = 1000119004,
// Provided by XR_FB_render_model
XR_TYPE_RENDER_MODEL_CAPABILITIES_REQUEST_FB = 1000119005,
// Provided by XR_KHR_binding_modification
XR_TYPE_BINDING_MODIFICATIONS_KHR = 1000120000,
// Provided by XR_VARJO_foveated_rendering
XR_TYPE_VIEW_LOCATE_FOVEATED_RENDERING_VARJO = 1000121000,
// Provided by XR_VARJO_foveated_rendering
XR_TYPE_FOVEATED_VIEW_CONFIGURATION_VIEW_VARJO = 1000121001,
// Provided by XR_VARJO_foveated_rendering
XR_TYPE_SYSTEM_FOVEATED_RENDERING_PROPERTIES_VARJO = 1000121002,
// Provided by XR_VARJO_composition_layer_depth_test
XR_TYPE_COMPOSITION_LAYER_DEPTH_TEST_VARJO = 1000122000,
// Provided by XR_VARJO_marker_tracking
XR_TYPE_SYSTEM_MARKER_TRACKING_PROPERTIES_VARJO = 1000124000,
// Provided by XR_VARJO_marker_tracking
XR_TYPE_EVENT_DATA_MARKER_TRACKING_UPDATE_VARJO = 1000124001,
// Provided by XR_VARJO_marker_tracking
XR_TYPE_MARKER_SPACE_CREATE_INFO_VARJO = 1000124002,
// Provided by XR_ML_frame_end_info
XR_TYPE_FRAME_END_INFO_ML = 1000135000,
// Provided by XR_ML_global_dimmer
XR_TYPE_GLOBAL_DIMMER_FRAME_END_INFO_ML = 1000136000,
// Provided by XR_ML_compat
XR_TYPE_COORDINATE_SPACE_CREATE_INFO_ML = 1000137000,

```

```

// Provided by XR_ML_marker_understanding
XR_TYPE_SYSTEM_MARKER_UNDERSTANDING_PROPERTIES_ML = 1000138000,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_CREATE_INFO_ML = 1000138001,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_ARUCO_INFO_ML = 1000138002,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_SIZE_INFO_ML = 1000138003,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_APRIL_TAG_INFO_ML = 1000138004,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_CUSTOM_PROFILE_INFO_ML = 1000138005,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_SNAPSHOT_INFO_ML = 1000138006,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_DETECTOR_STATE_ML = 1000138007,
// Provided by XR_ML_marker_understanding
XR_TYPE_MARKER_SPACE_CREATE_INFO_ML = 1000138008,
// Provided by XR_ML_localization_map
XR_TYPE_LOCALIZATION_MAP_ML = 1000139000,
// Provided by XR_ML_localization_map
XR_TYPE_EVENT_DATA_LOCALIZATION_CHANGED_ML = 1000139001,
// Provided by XR_ML_localization_map
XR_TYPE_MAP_LOCALIZATION_REQUEST_INFO_ML = 1000139002,
// Provided by XR_ML_localization_map
XR_TYPE_LOCALIZATION_MAP_IMPORT_INFO_ML = 1000139003,
// Provided by XR_ML_localization_map
XR_TYPE_LOCALIZATION_ENABLE_EVENTS_INFO_ML = 1000139004,
// Provided by XR_ML_user_calibration
XR_TYPE_EVENT_DATA_HEADSET_FIT_CHANGED_ML = 1000472000,
// Provided by XR_ML_user_calibration
XR_TYPE_EVENT_DATA_EYE_CALIBRATION_CHANGED_ML = 1000472001,
// Provided by XR_ML_user_calibration
XR_TYPE_USER_CALIBRATION_ENABLE_EVENTS_INFO_ML = 1000472002,
// Provided by XR_MSFT_spatial_anchor_persistence
XR_TYPE_SPATIAL_ANCHOR_PERSISTENCE_INFO_MSFT = 1000142000,
// Provided by XR_MSFT_spatial_anchor_persistence
XR_TYPE_SPATIAL_ANCHOR_FROM_PERSISTED_ANCHOR_CREATE_INFO_MSFT = 1000142001,
// Provided by XR_MSFT_scene_marker
XR_TYPE_SCENE_MARKERS_MSFT = 1000147000,
// Provided by XR_MSFT_scene_marker
XR_TYPE_SCENE_MARKER_TYPE_FILTER_MSFT = 1000147001,
// Provided by XR_MSFT_scene_marker
XR_TYPE_SCENE_MARKER_QR_CODES_MSFT = 1000147002,
// Provided by XR_FB_spatial_entity_query
XR_TYPE_SPACE_QUERY_INFO_FB = 1000156001,
// Provided by XR_FB_spatial_entity_query
XR_TYPE_SPACE_QUERY_RESULTS_FB = 1000156002,

```

```

// Provided by XR_FB_spatial_entity_query
XR_TYPE_SPACE_STORAGE_LOCATION_FILTER_INFO_FB = 1000156003,
// Provided by XR_FB_spatial_entity_query
XR_TYPE_SPACE_UUID_FILTER_INFO_FB = 1000156054,
// Provided by XR_FB_spatial_entity_query
XR_TYPE_SPACE_COMPONENT_FILTER_INFO_FB = 1000156052,
// Provided by XR_FB_spatial_entity_query
XR_TYPE_EVENT_DATA_SPACE_QUERY_RESULTS_AVAILABLE_FB = 1000156103,
// Provided by XR_FB_spatial_entity_query
XR_TYPE_EVENT_DATA_SPACE_QUERY_COMPLETE_FB = 1000156104,
// Provided by XR_FB_spatial_entity_storage
XR_TYPE_SPACE_SAVE_INFO_FB = 1000158000,
// Provided by XR_FB_spatial_entity_storage
XR_TYPE_SPACE_ERASE_INFO_FB = 1000158001,
// Provided by XR_FB_spatial_entity_storage
XR_TYPE_EVENT_DATA_SPACE_SAVE_COMPLETE_FB = 1000158106,
// Provided by XR_FB_spatial_entity_storage
XR_TYPE_EVENT_DATA_SPACE_ERASE_COMPLETE_FB = 1000158107,
// Provided by XR_FB_foveation_vulkan
XR_TYPE_SWAPCHAIN_IMAGE_FOVEATION_VULKAN_FB = 1000160000,
// Provided by XR_FB_swapchain_update_state_android_surface
XR_TYPE_SWAPCHAIN_STATE_ANDROID_SURFACE_DIMENSIONS_FB = 1000161000,
// Provided by XR_FB_swapchain_update_state_opengl_es
XR_TYPE_SWAPCHAIN_STATE_SAMPLER_OPENGL_ES_FB = 1000162000,
// Provided by XR_FB_swapchain_update_state_vulkan
XR_TYPE_SWAPCHAIN_STATE_SAMPLER_VULKAN_FB = 1000163000,
// Provided by XR_FB_spatial_entity_sharing
XR_TYPE_SPACE_SHARE_INFO_FB = 1000169001,
// Provided by XR_FB_spatial_entity_sharing
XR_TYPE_EVENT_DATA_SPACE_SHARE_COMPLETE_FB = 1000169002,
// Provided by XR_FB_space_warp
XR_TYPE_COMPOSITION_LAYER_SPACE_WARP_INFO_FB = 1000171000,
// Provided by XR_FB_space_warp
XR_TYPE_SYSTEM_SPACE_WARP_PROPERTIES_FB = 1000171001,
// Provided by XR_FB_haptic_amplitude_envelope
XR_TYPE_HAPTIC_AMPLITUDE_ENVELOPE_VIBRATION_FB = 1000173001,
// Provided by XR_FB_scene
XR_TYPE_SEMANTIC_LABELS_FB = 1000175000,
// Provided by XR_FB_scene
XR_TYPE_ROOM_LAYOUT_FB = 1000175001,
// Provided by XR_FB_scene
XR_TYPE_BOUNDARY_2D_FB = 1000175002,
// Provided by XR_FB_scene
XR_TYPE_SEMANTIC_LABELS_SUPPORT_INFO_FB = 1000175010,
// Provided by XR_ALMALENCE_digital_lens_control
XR_TYPE_DIGITAL_LENS_CONTROL_ALMALENCE = 1000196000,
// Provided by XR_FB_scene_capture
XR_TYPE_EVENT_DATA_SCENE_CAPTURE_COMPLETE_FB = 1000198001,

```

```

// Provided by XR_FB_scene_capture
XR_TYPE_SCENE_CAPTURE_REQUEST_INFO_FB = 1000198050,
// Provided by XR_FB_spatial_entity_container
XR_TYPE_SPACE_CONTAINER_FB = 1000199000,
// Provided by XR_META_foveation_eye_tracked
XR_TYPE_FOVEATION_EYE_TRACKED_PROFILE_CREATE_INFO_META = 1000200000,
// Provided by XR_META_foveation_eye_tracked
XR_TYPE_FOVEATION_EYE_TRACKED_STATE_META = 1000200001,
// Provided by XR_META_foveation_eye_tracked
XR_TYPE_SYSTEM_FOVEATION_EYE_TRACKED_PROPERTIES_META = 1000200002,
// Provided by XR_FB_face_tracking
XR_TYPE_SYSTEM_FACE_TRACKING_PROPERTIES_FB = 1000201004,
// Provided by XR_FB_face_tracking
XR_TYPE_FACE_TRACKER_CREATE_INFO_FB = 1000201005,
// Provided by XR_FB_face_tracking
XR_TYPE_FACE_EXPRESSION_INFO_FB = 1000201002,
// Provided by XR_FB_face_tracking
XR_TYPE_FACE_EXPRESSION_WEIGHTS_FB = 1000201006,
// Provided by XR_FB_eye_tracking_social
XR_TYPE_EYE_TRACKER_CREATE_INFO_FB = 1000202001,
// Provided by XR_FB_eye_tracking_social
XR_TYPE_EYE_GAZES_INFO_FB = 1000202002,
// Provided by XR_FB_eye_tracking_social
XR_TYPE_EYE_GAZES_FB = 1000202003,
// Provided by XR_FB_eye_tracking_social
XR_TYPE_SYSTEM_EYE_TRACKING_PROPERTIES_FB = 1000202004,
// Provided by XR_FB_passthrough_keyboard_hands
XR_TYPE_PASSTHROUGH_KEYBOARD_HANDS_INTENSITY_FB = 1000203002,
// Provided by XR_FB_composition_layer_settings
XR_TYPE_COMPOSITION_LAYER_SETTINGS_FB = 1000204000,
// Provided by XR_FB_haptic_pcm
XR_TYPE_HAPTIC_PCM_VIBRATION_FB = 1000209001,
// Provided by XR_FB_haptic_pcm
XR_TYPE_DEVICE_PCM_SAMPLE_RATE_STATE_FB = 1000209002,
// Provided by XR_FB_composition_layer_depth_test
XR_TYPE_COMPOSITION_LAYER_DEPTH_TEST_FB = 1000212000,
// Provided by XR_META_local_dimming
XR_TYPE_LOCAL_DIMMING_FRAME_END_INFO_META = 1000216000,
// Provided by XR_META_passthrough_preferences
XR_TYPE_PASSTHROUGH_PREFERENCES_META = 1000217000,
// Provided by XR_META_virtual_keyboard
XR_TYPE_SYSTEM_VIRTUAL_KEYBOARD_PROPERTIES_META = 1000219001,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_CREATE_INFO_META = 1000219002,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_SPACE_CREATE_INFO_META = 1000219003,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_LOCATION_INFO_META = 1000219004,

```

```

// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_MODEL_VISIBILITY_SET_INFO_META = 1000219005,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_ANIMATION_STATE_META = 1000219006,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_MODEL_ANIMATION_STATES_META = 1000219007,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_TEXTURE_DATA_META = 1000219009,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_INPUT_INFO_META = 1000219010,
// Provided by XR_META_virtual_keyboard
XR_TYPE_VIRTUAL_KEYBOARD_TEXT_CONTEXT_CHANGE_INFO_META = 1000219011,
// Provided by XR_META_virtual_keyboard
XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_COMMIT_TEXT_META = 1000219014,
// Provided by XR_META_virtual_keyboard
XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_BACKSPACE_META = 1000219015,
// Provided by XR_META_virtual_keyboard
XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_ENTER_META = 1000219016,
// Provided by XR_META_virtual_keyboard
XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_SHOWN_META = 1000219017,
// Provided by XR_META_virtual_keyboard
XR_TYPE_EVENT_DATA_VIRTUAL_KEYBOARD_HIDDEN_META = 1000219018,
// Provided by XR_OCULUS_external_camera
XR_TYPE_EXTERNAL_CAMERA_OCULUS = 1000226000,
// Provided by XR_META_vulkan_swapchain_create_info
XR_TYPE_VULKAN_SWAPCHAIN_CREATE_INFO_META = 1000227000,
// Provided by XR_META_performance_metrics
XR_TYPE_PERFORMANCE_METRICS_STATE_META = 1000232001,
// Provided by XR_META_performance_metrics
XR_TYPE_PERFORMANCE_METRICS_COUNTER_META = 1000232002,
// Provided by XR_FB_spatial_entity_storage_batch
XR_TYPE_SPACE_LIST_SAVE_INFO_FB = 1000238000,
// Provided by XR_FB_spatial_entity_storage_batch
XR_TYPE_EVENT_DATA_SPACE_LIST_SAVE_COMPLETE_FB = 1000238001,
// Provided by XR_FB_spatial_entity_user
XR_TYPE_SPACE_USER_CREATE_INFO_FB = 1000241001,
// Provided by XR_META_headset_id
XR_TYPE_SYSTEM_HEADSET_ID_PROPERTIES_META = 1000245000,
// Provided by XR_META_recommended_layer_resolution
XR_TYPE_RECOMMENDED_LAYER_RESOLUTION_META = 1000254000,
// Provided by XR_META_recommended_layer_resolution
XR_TYPE_RECOMMENDED_LAYER_RESOLUTION_GET_INFO_META = 1000254001,
// Provided by XR_META_passthrough_color_lut
XR_TYPE_SYSTEM_PASSTHROUGH_COLOR_LUT_PROPERTIES_META = 1000266000,
// Provided by XR_META_passthrough_color_lut
XR_TYPE_PASSTHROUGH_COLOR_LUT_CREATE_INFO_META = 1000266001,
// Provided by XR_META_passthrough_color_lut
XR_TYPE_PASSTHROUGH_COLOR_LUT_UPDATE_INFO_META = 1000266002,

```



```

// Provided by XR_META_passthrough_color_lut
XR_TYPE_PASSTHROUGH_COLOR_MAP_LUT_META = 1000266100,
// Provided by XR_META_passthrough_color_lut
XR_TYPE_PASSTHROUGH_COLOR_MAP_INTERPOLATED_LUT_META = 1000266101,
// Provided by XR_META_spatial_entity_mesh
XR_TYPE_SPACE_TRIANGLE_MESH_GET_INFO_META = 1000269001,
// Provided by XR_META_spatial_entity_mesh
XR_TYPE_SPACE_TRIANGLE_MESH_META = 1000269002,
// Provided by XR_FB_face_tracking2
XR_TYPE_SYSTEM_FACE_TRACKING_PROPERTIES2_FB = 1000287013,
// Provided by XR_FB_face_tracking2
XR_TYPE_FACE_TRACKER_CREATE_INFO2_FB = 1000287014,
// Provided by XR_FB_face_tracking2
XR_TYPE_FACE_EXPRESSION_INFO2_FB = 1000287015,
// Provided by XR_FB_face_tracking2
XR_TYPE_FACE_EXPRESSION_WEIGHTS2_FB = 1000287016,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_PROVIDER_CREATE_INFO_META = 1000291000,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_SWAPCHAIN_CREATE_INFO_META = 1000291001,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_SWAPCHAIN_STATE_META = 1000291002,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_IMAGE_ACQUIRE_INFO_META = 1000291003,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_IMAGE_VIEW_META = 1000291004,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_IMAGE_META = 1000291005,
// Provided by XR_META_environment_depth
XR_TYPE_ENVIRONMENT_DEPTH_HAND_REMOVAL_SET_INFO_META = 1000291006,
// Provided by XR_META_environment_depth
XR_TYPE_SYSTEM_ENVIRONMENT_DEPTH_PROPERTIES_META = 1000291007,
// Provided by XR_HTC_passthrough
XR_TYPE_PASSTHROUGH_CREATE_INFO_HTC = 1000317001,
// Provided by XR_HTC_passthrough
XR_TYPE_PASSTHROUGH_COLOR_HTC = 1000317002,
// Provided by XR_HTC_passthrough
XR_TYPE_PASSTHROUGH_MESH_TRANSFORM_INFO_HTC = 1000317003,
// Provided by XR_HTC_passthrough
XR_TYPE_COMPOSITION_LAYER_PASSTHROUGH_HTC = 1000317004,
// Provided by XR_HTC_foveation
XR_TYPE_FOVEATION_APPLY_INFO_HTC = 1000318000,
// Provided by XR_HTC_foveation
XR_TYPE_FOVEATION_DYNAMIC_MODE_INFO_HTC = 1000318001,
// Provided by XR_HTC_foveation
XR_TYPE_FOVEATION_CUSTOM_MODE_INFO_HTC = 1000318002,
// Provided by XR_HTC_anchor
XR_TYPE_SYSTEM_ANCHOR_PROPERTIES_HTC = 1000319000,

```

```

// Provided by XR_HTC_anchor
XR_TYPE_SPATIAL_ANCHOR_CREATE_INFO_HTC = 1000319001,
// Provided by XR_EXT_active_action_set_priority
XR_TYPE_ACTIVE_ACTION_SET_PRIORITIES_EXT = 1000373000,
// Provided by XR_MNDX_force_feedback_curl
XR_TYPE_SYSTEM_FORCE_FEEDBACK_CURL_PROPERTIES_MNDX = 1000375000,
// Provided by XR_MNDX_force_feedback_curl
XR_TYPE_FORCE_FEEDBACK_CURL_APPLY_LOCATIONS_MNDX = 1000375001,
// Provided by XR_EXT_hand_tracking_data_source
XR_TYPE_HAND_TRACKING_DATA_SOURCE_INFO_EXT = 1000428000,
// Provided by XR_EXT_hand_tracking_data_source
XR_TYPE_HAND_TRACKING_DATA_SOURCE_STATE_EXT = 1000428001,
// Provided by XR_EXT_plane_detection
XR_TYPE_PLANE_DETECTOR_CREATE_INFO_EXT = 1000429001,
// Provided by XR_EXT_plane_detection
XR_TYPE_PLANE_DETECTOR_BEGIN_INFO_EXT = 1000429002,
// Provided by XR_EXT_plane_detection
XR_TYPE_PLANE_DETECTOR_GET_INFO_EXT = 1000429003,
// Provided by XR_EXT_plane_detection
XR_TYPE_PLANE_DETECTOR_LOCATIONS_EXT = 1000429004,
// Provided by XR_EXT_plane_detection
XR_TYPE_PLANE_DETECTOR_LOCATION_EXT = 1000429005,
// Provided by XR_EXT_plane_detection
XR_TYPE_PLANE_DETECTOR_POLYGON_BUFFER_EXT = 1000429006,
// Provided by XR_EXT_plane_detection
XR_TYPE_SYSTEM_PLANE_DETECTION_PROPERTIES_EXT = 1000429007,
// Provided by XR_EXT_future
XR_TYPE_FUTURE_CANCEL_INFO_EXT = 1000469000,
// Provided by XR_EXT_future
XR_TYPE_FUTURE_POLL_INFO_EXT = 1000469001,
// Provided by XR_EXT_future
XR_TYPE_FUTURE_COMPLETION_EXT = 1000469002,
// Provided by XR_EXT_future
XR_TYPE_FUTURE_POLL_RESULT_EXT = 1000469003,
// Provided by XR_EXT_user_presence
XR_TYPE_EVENT_DATA_USER_PRESENCE_CHANGED_EXT = 1000470000,
// Provided by XR_EXT_user_presence
XR_TYPE_SYSTEM_USER_PRESENCE_PROPERTIES_EXT = 1000470001,
// Provided by XR_KHR_vulkan_enable2
XR_TYPE_GRAPHICS_BINDING_VULKAN2_KHR = XR_TYPE_GRAPHICS_BINDING_VULKAN_KHR,
// Provided by XR_KHR_vulkan_enable2
XR_TYPE_SWAPCHAIN_IMAGE_VULKAN2_KHR = XR_TYPE_SWAPCHAIN_IMAGE_VULKAN_KHR,
// Provided by XR_KHR_vulkan_enable2
XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN2_KHR = XR_TYPE_GRAPHICS_REQUIREMENTS_VULKAN_KHR,
// Provided by XR_FB_haptic_pcm
XR_TYPE_DEVICE_PCM_SAMPLE_RATE_GET_INFO_FB = XR_TYPE_DEVICE_PCM_SAMPLE_RATE_STATE_FB,
// Provided by XR_KHR_locate_spaces
XR_TYPE_SPACES_LOCATE_INFO_KHR = XR_TYPE_SPACES_LOCATE_INFO,

```

```
// Provided by XR_KHR_locate_spaces
XR_TYPE_SPACE_LOCATIONS_KHR = XR_TYPE_SPACE_LOCATIONS,
// Provided by XR_KHR_locate_spaces
XR_TYPE_SPACE_VELOCITIES_KHR = XR_TYPE_SPACE_VELOCITIES,
XR_STRUCTURE_TYPE_MAX_ENUM = 0x7FFFFFFF
} XrStructureType;
```

Most structures containing **type** members have a value of **type** matching the type of the structure, as described more fully in [Valid Usage for Structure Types](#).

Note that all extension enums begin at the extension enum base of  $10^9$  (base 10). Each extension is assigned a block of 1000 enums, starting at the enum base and arranged by the extension's number.

```
// Provided by XR_VERSION_1_0
#define XR_EXTENSION_ENUM_BASE 1000000000
```

```
// Provided by XR_VERSION_1_0
#define XR_EXTENSION_ENUM_STRIDE 1000
```

For example, if extension number 5 wants to use an enum value of 3, the final enum is computed by:

$$\text{enum} = \text{XR\_EXTENSION\_ENUM\_BASE} + (\text{extension\_number} - 1) * \text{XR\_EXTENSION\_ENUM\_STRIDE} + \text{enum\_value}$$

$$1000004003 = 1000000000 + 4 * 1000 + 3$$

The maximum allowed enum value in an extension is 2,147,482,999, which belongs to extension number 2147483.

## Flag Types

Flag types are all bitmasks aliasing the base type [XrFlags64](#) and with corresponding bit flag types defining the valid bits for that flag, as described in [Valid Usage for Flags](#).

Flag types defined in the core specification were originally listed/defined here, but have been moved to be adjacent to their associated [FlagBits](#) type. See the Index for a list.

## General Macro Definitions

This API is defined in C and uses "C" linkage. The [openxr.h](#) header file is opened with:

```
1 #ifdef __cplusplus
2 extern "C" {
3 #endif
```

and closed with:

```
1 #ifdef __cplusplus
2 }
3 #endif
```

The supplied `openxr.h` header defines a small number of C preprocessor macros that are described below.

### Version Number Macros

Two version numbers are defined in `openxr.h`. Each is packed into a 32-bit integer as described in [API Version Number Function-like Macros](#).

```
// Provided by XR_VERSION_1_0
// OpenXR current version number.
#define XR_CURRENT_API_VERSION XR_MAKE_VERSION(1, 1, 36)
```

`XR_CURRENT_API_VERSION` is the current version of the OpenXR API.

### API Version Number Function-like Macros

API Version Numbers are three components, packed into a single 64-bit integer. The following macros manipulate version components and packed version numbers.

```
#define XR_MAKE_VERSION(major, minor, patch) \
    (((major) & 0xffffULL) << 48) | (((minor) & 0xffffULL) << 32) | ((patch) & \
    0xffffffffULL))
```

## Parameter Descriptions

- **major** is the major version number, packed into the most-significant 16 bits.
- **minor** is the minor version number, packed into the second-most-significant group of 16 bits.
- **patch** is the patch version number, in the least-significant 32 bits.

[XR\\_MAKE\\_VERSION](#) constructs a packed 64-bit integer API version number from three components. The format used is described in [API Version Numbers and Semantics](#).

This macro **can** be used when constructing the [XrApplicationInfo::apiVersion](#) parameter passed to [xrCreateInstance](#).

```
// Provided by XR_VERSION_1_0
#define XR_VERSION_MAJOR(version) (uint16_t)((((uint64_t)(version) >> 48) & 0xffffULL))
```

## Parameter Descriptions

- **version** is a packed version number, such as those produced with [XR\\_MAKE\\_VERSION](#).

[XR\\_VERSION\\_MAJOR](#) extracts the API major version number from a packed version number.

```
// Provided by XR_VERSION_1_0
#define XR_VERSION_MINOR(version) (uint16_t)((((uint64_t)(version) >> 32) & 0xffffULL))
```

## Parameter Descriptions

- **version** is a packed version number, such as those produced with [XR\\_MAKE\\_VERSION](#).

[XR\\_VERSION\\_MINOR](#) extracts the API minor version number from a packed version number.

```
// Provided by XR_VERSION_1_0
#define XR_VERSION_PATCH(version) (uint32_t)((uint64_t)(version) & 0xffffffffULL)
```

## Parameter Descriptions

- `version` is a packed version number, such as those produced with [XR\\_MAKE\\_VERSION](#).

[XR\\_VERSION\\_PATCH](#) extracts the API patch version number from a packed version number.

### Handle and Atom Macros

```
// Provided by XR_VERSION_1_0
#if !defined(XR_DEFINE_HANDLE)
#if (XR_PTR_SIZE == 8)
    #define XR_DEFINE_HANDLE(object) typedef struct object##_T* object;
#else
    #define XR_DEFINE_HANDLE(object) typedef uint64_t object;
#endif
#endif
```

## Parameter Descriptions

- `object` is the name of the resulting C type.

[XR\\_DEFINE\\_HANDLE](#) defines a handle type, which is an opaque 64 bit value, which **may** be implemented as an opaque, distinct pointer type on platforms with 64 bit pointers.

For further details, see [Handles](#).

```
// Provided by XR_VERSION_1_0
#if !defined(XR_NULL_HANDLE)
#if (XR_PTR_SIZE == 8) && XR_CPP_NULLPTR_SUPPORTED
    #define XR_NULL_HANDLE nullptr
#else
    #define XR_NULL_HANDLE 0
#endif
#endif
```

[XR\\_NULL\\_HANDLE](#) is a reserved value representing a non-valid object handle. It **may** be passed to and returned from API functions only when specifically allowed.

```
#if !defined(XR_DEFINE_ATOM)
    #define XR_DEFINE_ATOM(object) typedef uint64_t object;
#endif
```

### Parameter Descriptions

- object is the name of the resulting C type.

[XR\\_DEFINE\\_ATOM](#) defines an atom type, which is an opaque 64 bit integer.

```
// Provided by XR_VERSION_1_0
#if !defined(XR_DEFINE_OPAQUE_64)
    #if (XR_PTR_SIZE == 8)
        #define XR_DEFINE_OPAQUE_64(object) typedef struct object##_T* object;
    #else
        #define XR_DEFINE_OPAQUE_64(object) typedef uint64_t object;
    #endif
#endif
#endif
```

### Parameter Descriptions

- object is the name of the resulting C type.

[XR\\_DEFINE\\_OPAQUE\\_64](#) defines an opaque 64 bit value, which **may** be implemented as an opaque, distinct pointer type on platforms with 64 bit pointers.

## Platform-Specific Macro Definitions

Additional platform-specific macros and interfaces are defined using the included `openxr_platform.h` file. These macros are used to control platform-dependent behavior, and their exact definitions are under the control of specific platform implementations of the API.

### Platform-Specific Calling Conventions

On many platforms the following macros are empty strings, causing platform- and compiler-specific default calling conventions to be used.

[XR\\_API\\_ATTR](#) is a macro placed before the return type of an API function declaration. This macro controls calling conventions for C++11 and GCC/Clang-style compilers.

[XRAPI\\_CALL](#) is a macro placed after the return type of an API function declaration. This macro controls calling conventions for MSVC-style compilers.

[XRAPI\\_PTR](#) is a macro placed between the ( and \* in API function pointer declarations. This macro also controls calling conventions, and typically has the same definition as [XRAPI\\_ATTR](#) or [XRAPI\\_CALL](#), depending on the compiler.

Examples:

Function declaration:

```
XRAPI_ATTR <return_type> XRAPI_CALL <function_name>(<function_parameters>);
```

Function pointer type declaration:

```
typedef <return_type> (XRAPI_PTR *PFN_<function_name>)(<function_parameters>);
```

### Platform-Specific Header Control

If the [XR\\_NO\\_STDINT\\_H](#) macro is defined by the application at compile time, before including any OpenXR header, extended integer types normally found in [<stdint.h>](#) and used by the OpenXR headers, such as [uint8\\_t](#), **must** also be defined (as [typedef](#) or with the preprocessor) before including any OpenXR header. Otherwise, [openxr.h](#) and related headers will not compile. If [XR\\_NO\\_STDINT\\_H](#) is not defined, the system-provided [<stdint.h>](#) is used to define these types. There is a fallback path for Microsoft Visual Studio version 2008 and earlier versions (which lack this header) that is automatically activated as needed.

### Graphics API Header Control

| Compile Time Symbol                           | Graphics API Name |
|---|-------------------|
| <a href="#">XR_USE_GRAPHICS_API_OPENGL</a>    | OpenGL            |
| <a href="#">XR_USE_GRAPHICS_API_OPENGL_ES</a> | OpenGL ES         |
| <a href="#">XR_USE_GRAPHICS_API_VULKAN</a>    | Vulkan            |
| <a href="#">XR_USE_GRAPHICS_API_D3D11</a>     | Direct3D 11       |
| <a href="#">XR_USE_GRAPHICS_API_D3D12</a>     | Direct3D 12       |

### Window System Header Control

| Compile Time Symbol                   | Window System Name   |
|---------------------------------------|----------------------|
| <a href="#">XR_USE_PLATFORM_WIN32</a> | Microsoft Windows    |
| <a href="#">XR_USE_PLATFORM_XLIB</a>  | X Window System Xlib |



| Compile Time Symbol     | Window System Name  |
|-------------------------|---------------------|
| XR_USE_PLATFORM_XCB     | X Window System XCB |
| XR_USE_PLATFORM_WAYLAND | Wayland             |
| XR_USE_PLATFORM_ANDROID | Android Native      |

## Android Notes

Android specific notes for using the OpenXR specification.

### Android Runtime category tag for immersive mode selection

Android applications should add the `<category android:name="org.khronos.openxr.intent.category.IMMERSIVE_HMD" />` tag inside the intent-filter to indicate that the activity starts in an immersive OpenXR mode and will not touch the native Android 2D surface.

The HMD suffix indicates the preferred form-factor used by the application and can be used by launchers to filter applications listed.

For example:

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
  <category android:name="org.khronos.openxr.intent.category.IMMERSIVE_HMD" />
</intent-filter>
```

## Glossary

The terms defined in this section are used throughout this Specification. Capitalization is not significant for these definitions.

| Term        | Description  |
|-------------|--|
| Application | The XR application which calls the OpenXR API to communicate with an OpenXR runtime. |

| Term           | Description  |
|----------------|--|
| Deprecated     | A feature/extension is deprecated if it is no longer recommended as the correct or best way to achieve its intended purpose. Generally a newer feature/extension will have been created that solves the same problem - in cases where no newer alternative feature exists, justification should be provided.   |
| Handle         | An opaque integer or pointer value used to refer to an object. Each object type has a unique handle type.  |
| Haptic         | Haptic or kinesthetic communication recreates the sense of touch by applying forces, vibrations, or motions to the user.   |
| In-Process     | Something that executes in the application's process.  |
| Instance       | The top-level object, which represents the application's connection to the runtime. Represented by an <a href="#">XrInstance</a> object.   |
| Normalized     | A value that is interpreted as being in the range [0,1], or a vector whose norm is in that range, as a result of being implicitly divided or scaled by some other value.   |
| Out-Of-Process | Something that executes outside the application's process.   |
| Promoted       | <p>A feature is promoted if it is taken from an older extension and made available as part of a new core version of the API, or a newer extension that is considered to be either as widely supported or more so. A promoted feature may have minor differences from the original such as:</p> <ul style="list-style-type: none"> <li>• It may be renamed</li> <li>• A small number of non-intrusive parameters may have been added</li> <li>• The feature may be advertised differently by device features</li> <li>• The author ID suffixes will be changed or removed as appropriate</li> </ul> |

| <b>Term</b>         | <b>Description</b>   |
|---------------------|--|
| Provisional         | A feature is released provisionally in order to get wider feedback on the functionality before it is finalized. Provisional features may change in ways that break backwards compatibility, and thus are not recommended for use in production applications. |
| Required Extensions | Extensions that must be enabled alongside extensions dependent on them, or that must be enabled to use given hardware.   |
| Runtime             | The software which implements the OpenXR API and allows applications to interact with XR hardware.   |
| Swapchain           | A resource that represents a chain of images in device memory. Represented by an <a href="#">XrSwapchain</a> object.   |
| Swapchain Image     | Each element in a swapchain. Commonly these are simple formatted 2D images, but in other cases they may be array images. Represented by a structure related to <a href="#">XrSwapchainImageBaseHeader</a> .  |

## Abbreviations

Abbreviations and acronyms are sometimes used in the API where they are considered clear and commonplace, and are defined here:

| <b>Abbreviation</b> | <b>Description</b>  |
|---------------------|---|
| API                 | Application Programming Interface   |
| AR                  | Augmented Reality   |
| ER                  | Eye Relief  |
| IAD                 | Inter Axial Distance  |
| IPD                 | Inter Pupillary Distance  |
| MR                  | Mixed Reality   |
| OS                  | Operating System  |
| TSG                 | Technical Sub-Group. A specialized sub-group within a Khronos Working Group (WG). |
| VR                  | Virtual Reality   |

| <b>Abbreviation</b> | <b>Description</b>  |
|---------------------|---|
| WG                  | Working Group. An organized group of people working to define/augment an API. |
| XR                  | VR + AR + MR  |

# Dedication (Informative)

In memory of Johannes van Waveren: a loving father, husband, son, brother, colleague, and dear friend.

Johannes, known to his friends as "JP", had a great sense of humor, fierce loyalty, intense drive, a love of rainbow unicorns, and deep disdain for processed American cheese. Perhaps most distinguishing of all, though, was his love of technology and his extraordinary technical ability.

JP's love of technology started at an early age --- instead of working on his homework, he built train sets, hovercrafts, and complex erector sets from scratch; fashioned a tool for grabbing loose change out of street grates; and played computer games. The passion for computer games continued at Delft University of Technology, where, armed with a T1 internet connection and sheer talent, he regularly destroyed his foes in arena matches without being seen, earning him the moniker "MrElusive". During this time, he wrote the Gladiator-bot AI, which earned him acclaim in the community and led directly to a job at the iconic American computer game company, id Software. From there, he quickly became an expert in every system he touched, contributing significantly to every facet of the technology: AI, path navigation, networking, skeletal animation, virtual texturing, advanced rendering, and physics. He became a master of all. He famously owned more lines of code than anyone else, but he was also a generous mentor, helping junior developers hone their skills and make their own contributions.

When the chance to work in the VR industry arose, he saw it as an opportunity to help shape the future. Having never worked on VR hardware did not phase him; he quickly became a top expert in the field. Many of his contributions directly moved the industry forward, most recently his work on asynchronous timewarp and open-standards development.

Time was not on his side. Even in his final days, JP worked tirelessly on the initial proposal for this specification. The treatments he had undergone took a tremendous physical toll, but he continued to work because of his love of technology, his dedication to the craft, and his desire to get OpenXR started on a solid footing. His focus was unwavering.

His proposal was unofficially adopted several days before his passing - and upon hearing, he mustered the energy for a smile. While it was his great dream to see this process through, he would be proud of the spirit of cooperation, passion, and dedication of the industry peers who took up the torch to drive this specification to completion.

JP lived a life full of accomplishment, as evidenced by many publications, credits, awards, and nominations where you will find his name. A less obvious accomplishment --- but of equal importance --- is the influence he had on people through his passionate leadership. He strove for excellence in everything that he did. He was always excited to talk about technology and share the discoveries made while working through complex problems. He created excitement and interest around engineering and technical excellence. He was a mentor and teacher who inspired those who knew him and many continue to benefit from his hard work and generosity.

JP was a rare gem; fantastically brilliant intellectually, but also warm, compassionate, generous, humble, and funny. Those of us lucky enough to have crossed paths with him knew what a privilege and great honor it was to know him. He is certainly missed.



# Contributors (Informative)

OpenXR is the result of contributions from many people and companies participating in the Khronos OpenXR Working Group. Members of the Working Group, including the company that they represented at the time of their most recent contribution, are listed below.

## Working Group Contributors to OpenXR

- Adam Gousetis, Google (version 1.0)
- Alain Zanchetta, Microsoft (version 1.1)
- Alex Turner, Microsoft (versions 1.0, 1.1)
- Alex Sink, HTC (version 1.1)
- Alfredo Muniz, XEED (version 1.1) (Working Group Chair)
- Andreas Loeve Selvik, Meta Platforms (versions 1.0, 1.1)
- Andres Rodriguez, Valve Software (version 1.0)
- Armelle Laine, Qualcomm Technologies (version 1.0)
- Attila Maczak, CTRL-labs (version 1.0)
- David Fields, Microsoft (version 1.1)
- Baolin Fu, Bytedance (version 1.1)
- Blake Taylor, Magic Leap (version 1.0)
- Brad Grantham, Google (version 1.0)
- Brandon Jones, Google (version 1.0)
- Brent E. Insko, Intel (version 1.0) (former Working Group Chair)
- Brent Wilson, Microsoft (version 1.0)
- Bryce Hutchings, Microsoft (versions 1.0, 1.1)
- Cass Everitt, Meta Platforms (versions 1.0, 1.1)
- Charles Egenbacher, Epic Games (version 1.0)
- Charlton Rodda, Collabora (version 1.1)
- Chris Kuo, HTC (version 1.1)
- Chris Osborn, CTRL-labs (version 1.0)
- Christine Perey, Perey Research & Consulting (version 1.0)
- Christoph Haag, Collabora (version 1.0, 1.1)
- Christopher Fiala, Epic Games (version 1.1)
- Craig Donner, Google (version 1.0)

- Dan Ginsburg, Valve Software (version 1.0)
- Dave Houlton, LunarG (version 1.0)
- Dave Shreiner, Unity Technologies (version 1.0)
- Darryl Gough, Microsoft (version 1.1)
- Denny Rönngren, Varjo (versions 1.0, 1.1)
- Dmitriy Vasilev, Samsung Electronics (version 1.0)
- Doug Twileager, ZSpace (version 1.0)
- Ed Hutchins, Meta Platforms (version 1.0)
- Eryk Pecyna, Meta Platforms (version 1.1)
- Frederic Plourde, Collabora (version 1.1)
- Gloria Kennickell, Meta Platforms (version 1.0)
- Gregory Greeby, AMD (version 1.0)
- Guodong Chen, Huawei (version 1.0)
- Jack Pritz, Unity Technologies (versions 1.0, 1.1)
- Jakob Bornecrantz, Collabora (versions 1.0, 1.1)
- Jared Cheshier, PlutoVR (versions 1.0, 1.1)
- Jared Finder, Google (version 1.1)
- Javier Martinez, Intel (version 1.0)
- Jeff Bellinghausen, Valve Software (version 1.0)
- Jiehua Guo, Huawei (version 1.0)
- Joe Ludwig, Valve Software (versions 1.0, 1.1)
- John Kearney, Meta Platforms (version 1.1)
- Johannes van Waveren, Meta Platforms (version 1.0)
- Jon Leech, Khronos (version 1.0)
- Jonas Pegerfalk, Tobii (version 1.1)
- Jonathan Wright, Meta Platforms (versions 1.0, 1.1)
- Juan Wee, Samsung Electronics (version 1.0)
- Jules Blok, Epic Games (version 1.0)
- Jun Yan, ByteDance (version 1.1)
- Karl Schultz, LunarG (version 1.0)
- Karthik Kadappan, Magic Leap (version 1.1)
- Karthik Nagarajan, Qualcomm Technologies (version 1.1)



- Kaye Mason, Google (version 1.0)
- Krzysztof Kosiński, Google (version 1.0)
- Kyle Chen, HTC (version 1.1)
- Lachlan Ford, Google (versions 1.0, 1.1)
- Lubosz Sarnecki, Collabora (version 1.0)
- Mark Young, LunarG (version 1.0)
- Martin Renschler, Qualcomm Technologies (version 1.0)
- Matias Koskela, Tampere University of Technology (version 1.0)
- Matt Wash, Arm (version 1.0)
- Mattias Brand, Tobii (version 1.0)
- Mattias O. Karlsson, Tobii (version 1.0)
- Matthieu Bucchianeri, Microsoft (version 1.1)
- Michael Gatson, Dell (version 1.0)
- Minmin Gong, Microsoft (version 1.0)
- Mitch Singer, AMD (version 1.0)
- Nathan Nuber, Valve (version 1.1)
- Nell Waliczek, Microsoft (version 1.0)
- Nick Whiting, Epic Games (version 1.0) (former Working Group Chair)
- Nigel Williams, Sony (version 1.0)
- Nihav Jain, Google, Inc (version 1.1)
- Paul Pedriana, Meta Platforms (version 1.0)
- Paulo Gomes, Samsung Electronics (version 1.0)
- Peter Kuhn, Unity Technologies (versions 1.0, 1.1)
- Peter Peterson, HP Inc (version 1.0)
- Philippe Harscoet, Samsung Electronics (versions 1.0, 1.1)
- Pierre-Loup Griffais, Valve Software (version 1.0)
- Rafael Wiltz, Magic Leap (version 1.1)
- Rajeev Gupta, Sony (version 1.0)
- Remi Arnaud, Starbreeze (version 1.0)
- Remy Zimmerman, Logitech (version 1.0)
- Ria Hsu, HTC (version 1.1)
- River Gillis, Google (version 1.0)

- Robert Blenkinsopp, Ultraleap (version 1.1)
- Robert Memmott, Meta Platforms (version 1.0)
- Robert Menzel, NVIDIA (version 1.0)
- Robert Simpson, Qualcomm Technologies (version 1.0)
- Robin Bourianes, Starbreeze (version 1.0)
- Ron Bessems, Magic Leap (version 1.1) (Working Group Vice-Chair)
- Rune Berg, independent (version 1.1)
- Rylie Pavlik, Collabora (versions 1.0, 1.1) (Spec Editor)
- Ryan Vance, Epic Games (version 1.0)
- Sam Martin, Arm (version 1.0)
- Satish Salian, NVIDIA (version 1.0)
- Scott Flynn, Unity Technologies (version 1.0)
- Shanliang Xu, Bytedance (version 1.1)
- Sean Payne, CTRL-labs (version 1.0)
- Sophia Baldonado, PlutoVR (version 1.0)
- Steve Smith, Epic Games (version 1.0)
- Sungye Kim, Intel (version 1.0)
- Tom Flynn, Samsung Electronics (version 1.0)
- Trevor F. Smith, Mozilla (version 1.0)
- Victor Brodin, Epic Games (version 1.1)
- Vivek Viswanathan, Dell (version 1.0)
- Wenlin Mao, Meta Platforms (version 1.1)
- Xiang Wei, Meta Platforms (version 1.1)
- Yin Li, Microsoft (versions 1.0, 1.1)
- Yuval Boger, Sensics (version 1.0)
- Zhanrui Jia, Bytedance (version 1.1)
- Zheng Qin, Microsoft (version 1.0)

# Index

- A**
- `XR_API_LAYER_CREATE_INFO_STRUCT_VERSION` (define), [72](#)
  - `XR_API_LAYER_INFO_STRUCT_VERSION` (define), [69](#)
  - `XR_API_LAYER_MAX_SETTINGS_PATH_SIZE` (define), [72](#)
  - `XR_API_LAYER_NEXT_INFO_STRUCT_VERSION` (define), [73](#)
  - `xrAcquireEnvironmentDepthImageMETA` (function), [991](#)
  - `xrAcquireSwapchainImage` (function), [213](#)
  - `XrAction` (type), [255](#)
  - `XrActionCreateInfo` (type), [256](#)
  - `XrActionSet` (type), [251](#)
  - `XrActionSetCreateInfo` (type), [252](#)
  - `XrActionSpaceCreateInfo` (type), [157](#)
  - `XrActionsSyncInfo` (type), [289](#)
  - `XrActionStateBoolean` (type), [274](#)
  - `XrActionStateFloat` (type), [276](#)
  - `XrActionStateGetInfo` (type), [271](#)
  - `XrActionStatePose` (type), [281](#)
  - `XrActionStateVector2f` (type), [278](#)
  - `XrActionSuggestedBinding` (type), [264](#)
  - `XrActionType` (type), [259](#)
  - `XrActiveActionSet` (type), [289](#)
  - `XrActiveActionSetPrioritiesEXT` (type), [420](#)
  - `XrActiveActionSetPriorityEXT` (type), [422](#)
  - `XrAndroidSurfaceSwapchainCreateInfoFB` (type), [593](#)
  - `XrAndroidSurfaceSwapchainFlagBitsFB` (type), [592](#)
  - `XrAndroidSurfaceSwapchainFlagsFB` (type), [592](#)
  - `XrAndroidThreadTypeKHR` (type), [308](#)
  - `XrApiLayerCreateInfo` (type), [71](#)
  - `XrApiLayerNextInfo` (type), [72](#)
  - `XrApiLayerProperties` (type), [76](#)
  - `XrApplicationInfo` (type), [83](#)
  - `xrApplyForceFeedbackCurlMNDX` (function), [1418](#)
  - `xrApplyFoveationHTC` (function), [948](#)
  - `xrApplyHapticFeedback` (function), [282](#)
  - `XrAsyncRequestIdFB` (type), [822](#)
  - `xrAttachSessionActionSets` (function), [264](#)
- B**
- `XrBaseInStructure` (type), [17](#)
  - `XrBaseOutStructure` (type), [17](#)
  - `xrBeginFrame` (function), [230](#)
  - `xrBeginPlaneDetectionEXT` (function), [551](#)
  - `xrBeginSession` (function), [192](#)
  - `XrBindingModificationBaseHeaderKHR` (type), [313](#)
  - `XrBindingModificationsKHR` (type), [312](#)
  - `XrBlendFactorFB` (type), [620](#)
  - `XrBodyJointFB` (type), [609](#)
  - `XrBodyJointLocationFB` (type), [604](#)
  - `XrBodyJointLocationsFB` (type), [602](#)
  - `XrBodyJointSetFB` (type), [599](#)
  - `XrBodyJointsLocateInfoFB` (type), [601](#)
  - `XrBodySkeletonFB` (type), [606](#)
  - `XrBodySkeletonJointFB` (type), [606](#)
  - `XrBodyTrackerCreateInfoFB` (type), [598](#)
  - `XrBodyTrackerFB` (type), [596](#)
  - `XrBool32` (type), [49](#)
  - `XrBoundary2DFB` (type), [802](#)
  - `XrBoundSourcesForActionEnumerateInfo` (type), [292](#)
  - `XrBoxf` (type), [46](#)
  - `XrBoxfKHR` (type), [1434](#)
- C**
- `XR_CURRENT_API_VERSION` (define), [1492](#)
  - `XR_CURRENT_LOADER_API_LAYER_VERSION` (define), [63](#)
  - `XR_CURRENT_LOADER_RUNTIME_VERSION` (define), [63](#)
  - `xrCancelFutureEXT` (function), [480](#)
  - `xrChangeVirtualKeyboardTextContextMETA` (function), [1074](#)
  - `xrClearSpatialAnchorStoreMSFT` (function), [1321](#)
  - `XrColor3f` (type), [38](#)
  - `XrColor3fKHR` (type), [1433](#)
  - `XrColor4f` (type), [38](#)
  - `XrColorSpaceFB` (type), [613](#)
  - `XrCompareOpFB` (type), [623](#)

XrCompositionLayerAlphaBlendFB (type), 620  
 XrCompositionLayerBaseHeader (type), 238  
 XrCompositionLayerColorScaleBiasKHR (type), 315  
 XrCompositionLayerCubeKHR (type), 318  
 XrCompositionLayerCylinderKHR (type), 322  
 XrCompositionLayerDepthInfoKHR (type), 325  
 XrCompositionLayerDepthTestFB (type), 623  
 XrCompositionLayerDepthTestVARJO (type), 1361  
 XrCompositionLayerEquirect2KHR (type), 333  
 XrCompositionLayerEquirectKHR (type), 330  
 XrCompositionLayerFlagBits (type), 236  
 XrCompositionLayerFlags (type), 236  
 XrCompositionLayerImageLayoutFB (type), 626  
 XrCompositionLayerImageLayoutFlagBitsFB (type), 625  
 XrCompositionLayerImageLayoutFlagsFB (type), 625  
 XrCompositionLayerPassthroughFB (type), 757  
 XrCompositionLayerPassthroughHTC (type), 962  
 XrCompositionLayerProjection (type), 240  
 XrCompositionLayerProjectionView (type), 241  
 XrCompositionLayerQuad (type), 242  
 XrCompositionLayerReprojectionInfoMSFT (type), 1166  
     XrCompositionLayerReprojectionPlaneOverrideMSFT (type), 1168  
 XrCompositionLayerSecureContentFB (type), 629  
 XrCompositionLayerSecureContentFlagBitsFB (type), 628  
 XrCompositionLayerSecureContentFlagsFB (type), 628  
 XrCompositionLayerSettingsFB (type), 632  
 XrCompositionLayerSettingsFlagBitsFB (type), 630  
 XrCompositionLayerSettingsFlagsFB (type), 630  
 XrCompositionLayerSpaceWarpInfoFB (type), 817  
 XrCompositionLayerSpaceWarpInfoFlagBitsFB (type), 817  
 XrCompositionLayerSpaceWarpInfoFlagsFB (type), 817  
 xrComputeNewSceneMSFT (function), 1237  
 XrControllerModelKeyMSFT (type), 1173  
 XrControllerModelKeyStateMSFT (type), 1172  
 XrControllerModelNodePropertiesMSFT (type), 1178  
 XrControllerModelNodeStateMSFT (type), 1181  
 XrControllerModelPropertiesMSFT (type), 1177  
 XrControllerModelStateMSFT (type), 1181  
 xrConvertTimespecTimeToTimeKHR (function), 336  
 xrConvertTimeToTimespecTimeKHR (function), 337  
 xrConvertTimeToWin32PerformanceCounterKHR (function), 418  
 xrConvertWin32PerformanceCounterToTimeKHR (function), 417  
 XrCoordinateSpaceCreateInfoML (type), 1091  
 xrCreateAction (function), 255  
 xrCreateActionSet (function), 251  
 xrCreateActionSpace (function), 156  
 xrCreateApiLayerInstance (function), 69  
 xrCreateBodyTrackerFB (function), 596  
 xrCreateDebugUtilsMessengerEXT (function), 443  
 xrCreateEnvironmentDepthProviderMETA (function), 975  
 xrCreateEnvironmentDepthSwapchainMETA (function), 983  
 xrCreateExportedLocalizationMapML (function), 1112  
 xrCreateEyeTrackerFB (function), 642  
 xrCreateFaceTracker2FB (function), 669  
 xrCreateFaceTrackerFB (function), 654  
 xrCreateFacialTrackerHTC (function), 923  
 xrCreateFoveationProfileFB (function), 711  
 xrCreateGeometryInstanceFB (function), 774  
 xrCreateHandMeshSpaceMSFT (function), 1190  
 xrCreateHandTrackerEXT (function), 511  
 xrCreateInstance (function), 79  
 xrCreateKeyboardSpaceFB (function), 747  
 xrCreateMarkerDetectorML (function), 1120  
 xrCreateMarkerSpaceML (function), 1149  
 xrCreateMarkerSpaceVARJO (function), 1379  
 xrCreatePassthroughColorLutMETA (function), 1017  
 xrCreatePassthroughFB (function), 765  
 xrCreatePassthroughHTC (function), 959  
 xrCreatePassthroughLayerFB (function), 769

[xrCreatePlaneDetectorEXT \(function\)](#), [547](#)  
[xrCreateReferenceSpace \(function\)](#), [154](#)  
[xrCreateSceneMSFT \(function\)](#), [1251](#)  
[xrCreateSceneObserverMSFT \(function\)](#), [1233](#)  
[xrCreateSession \(function\)](#), [188](#)  
[xrCreateSpaceFromCoordinateFrameUIDML \(function\)](#), [1092](#)  
[xrCreateSpaceUserFB \(function\)](#), [876](#)  
[xrCreateSpatialAnchorFB \(function\)](#), [828](#)  
     [xrCreateSpatialAnchorFromPerceptionAnchorMSFT \(function\)](#), [1213](#)  
[xrCreateSpatialAnchorFromPersistedNameMSFT \(function\)](#), [1318](#)  
[xrCreateSpatialAnchorHTC \(function\)](#), [916](#)  
[xrCreateSpatialAnchorMSFT \(function\)](#), [1304](#)  
[xrCreateSpatialAnchorSpaceMSFT \(function\)](#), [1306](#)  
[xrCreateSpatialAnchorStoreConnectionMSFT \(function\)](#), [1311](#)  
[xrCreateSpatialGraphNodeSpaceMSFT \(function\)](#), [1324](#)  
[xrCreateSwapchain \(function\)](#), [204](#)  
[xrCreateSwapchainAndroidSurfaceKHR \(function\)](#), [305](#)  
[xrCreateTriangleMeshFB \(function\)](#), [904](#)  
[xrCreateVirtualKeyboardMETA \(function\)](#), [1051](#)  
[xrCreateVirtualKeyboardSpaceMETA \(function\)](#), [1054](#)  
[xrCreateVulkanDeviceKHR \(function\)](#), [404](#)  
[xrCreateVulkanInstanceKHR \(function\)](#), [399](#)

**D**

[XR\\_DEFINE\\_ATOM \(define\)](#), [1494](#)  
[XR\\_DEFINE\\_HANDLE \(define\)](#), [1494](#)  
[XR\\_DEFINE\\_OPAQUE\\_64 \(define\)](#), [1495](#)  
[XrDebugUtilsLabelEXT \(type\)](#), [438](#)  
[XrDebugUtilsMessageSeverityFlagBitsEXT \(type\)](#), [436](#)  
[XrDebugUtilsMessageSeverityFlagsEXT \(type\)](#), [436](#)  
[XrDebugUtilsMessageTypeFlagBitsEXT \(type\)](#), [436](#)  
[XrDebugUtilsMessageTypeFlagsEXT \(type\)](#), [436](#)  
[XrDebugUtilsMessengerCallbackDataEXT \(type\)](#), [438](#)  
[XrDebugUtilsMessengerCreateInfoEXT \(type\)](#), [440](#)  
[XrDebugUtilsMessengerEXT \(type\)](#), [435](#)  
[XrDebugUtilsObjectNameInfoEXT \(type\)](#), [437](#)  
[XrDeserializeSceneFragmentMSFT \(type\)](#), [1290](#)  
[xrDeserializeSceneMSFT \(function\)](#), [1287](#)  
[xrDestroyAction \(function\)](#), [259](#)  
[xrDestroyActionSet \(function\)](#), [254](#)  
[xrDestroyBodyTrackerFB \(function\)](#), [599](#)  
[xrDestroyDebugUtilsMessengerEXT \(function\)](#), [445](#)  
[xrDestroyEnvironmentDepthProviderMETA \(function\)](#), [977](#)  
[xrDestroyEnvironmentDepthSwapchainMETA \(function\)](#), [988](#)  
[xrDestroyExportedLocalizationMapML \(function\)](#), [1113](#)  
[xrDestroyEyeTrackerFB \(function\)](#), [644](#)  
[xrDestroyFaceTracker2FB \(function\)](#), [673](#)  
[xrDestroyFaceTrackerFB \(function\)](#), [656](#)  
[xrDestroyFacialTrackerHTC \(function\)](#), [926](#)  
[xrDestroyFoveationProfileFB \(function\)](#), [712](#)  
[xrDestroyGeometryInstanceFB \(function\)](#), [776](#)  
[xrDestroyHandTrackerEXT \(function\)](#), [514](#)  
[xrDestroyInstance \(function\)](#), [84](#)  
[xrDestroyMarkerDetectorML \(function\)](#), [1130](#)  
[xrDestroyPassthroughColorLutMETA \(function\)](#), [1018](#)  
[xrDestroyPassthroughFB \(function\)](#), [766](#)  
[xrDestroyPassthroughHTC \(function\)](#), [961](#)  
[xrDestroyPassthroughLayerFB \(function\)](#), [770](#)  
[xrDestroyPlaneDetectorEXT \(function\)](#), [550](#)  
[xrDestroySceneMSFT \(function\)](#), [1253](#)  
[xrDestroySceneObserverMSFT \(function\)](#), [1235](#)  
[xrDestroySession \(function\)](#), [191](#)  
[xrDestroySpace \(function\)](#), [158](#)  
[xrDestroySpaceUserFB \(function\)](#), [878](#)  
[xrDestroySpatialAnchorMSFT \(function\)](#), [1308](#)  
[xrDestroySpatialAnchorStoreConnectionMSFT \(function\)](#), [1312](#)  
[xrDestroySpatialGraphNodeBindingMSFT \(function\)](#), [1330](#)  
[xrDestroySwapchain \(function\)](#), [210](#)  
[xrDestroyTriangleMeshFB \(function\)](#), [905](#)  
[xrDestroyVirtualKeyboardMETA \(function\)](#), [1053](#)

XrDevicePcmSampleRateStateFB (type), [739](#)  
 XrDigitalLensControlALMALENCE (type), [587](#)  
 XrDigitalLensControlFlagBitsALMALENCE (type),  
[587](#)  
 XrDigitalLensControlFlagsALMALENCE (type), [587](#)  
 XrDuration (type), [37](#)

**E**

XR\_EXTENSION\_ENUM\_BASE (define), [1491](#)  
 XR\_EXTENSION\_ENUM\_STRIDE (define), [1491](#)  
 xrEnableLocalizationEventsML (function), [1102](#)  
 xrEnableUserCalibrationEventsML (function),  
[1158](#)  
 xrEndFrame (function), [232](#)  
 xrEndSession (function), [195](#)  
 xrEnumerateApiLayerProperties (function), [75](#)  
 xrEnumerateBoundSourcesForAction (function),  
[290](#)  
 xrEnumerateColorSpacesFB (function), [615](#)  
 xrEnumerateDisplayRefreshRatesFB (function),  
[635](#)  
 xrEnumerateEnvironmentBlendModes (function),  
[245](#)  
     xrEnumerateEnvironmentDepthSwapchainI  
     imagesMETA (function), [989](#)  
 xrEnumerateExternalCamerasOCULUS (function),  
[1345](#)  
 xrEnumerateInstanceExtensionProperties  
 (function), [77](#)  
     xrEnumeratePerformanceMetricsCounterPa  
     thsMETA (function), [1030](#)  
 xrEnumeratePersistedSpatialAnchorNamesMSFT  
 (function), [1316](#)  
 xrEnumerateReferenceSpaces (function), [152](#)  
 xrEnumerateRenderModelPathsFB (function), [790](#)  
 xrEnumerateReprojectionModesMSFT (function),  
[1165](#)  
 xrEnumerateSceneComputeFeaturesMSFT  
 (function), [1247](#)  
 xrEnumerateSpaceSupportedComponentsFB  
 (function), [831](#)  
 xrEnumerateSwapchainFormats (function), [202](#)  
 xrEnumerateSwapchainImages (function), [211](#)  
 xrEnumerateViewConfigurations (function), [177](#)  
 xrEnumerateViewConfigurationViews (function),  
[180](#)  
 xrEnumerateViveTrackerPathsHTCX (function),  
[1406](#)  
 XrEnvironmentBlendMode (type), [247](#)  
 XrEnvironmentDepthHandRemovalSetInfoMETA  
 (type), [982](#)  
 XrEnvironmentDepthImageAcquireInfoMETA  
 (type), [993](#)  
 XrEnvironmentDepthImageMETA (type), [995](#)  
 XrEnvironmentDepthImageViewMETA (type), [994](#)  
     XrEnvironmentDepthProviderCreateFlagBit  
     sMETA (type), [977](#)  
 XrEnvironmentDepthProviderCreateFlagsMETA  
 (type), [977](#)  
 XrEnvironmentDepthProviderCreateInfoMETA  
 (type), [976](#)  
 XrEnvironmentDepthProviderMETA (type), [975](#)  
     XrEnvironmentDepthSwapchainCreateFlag  
     BitsMETA (type), [986](#)  
 XrEnvironmentDepthSwapchainCreateFlagsMETA  
 (type), [986](#)  
 XrEnvironmentDepthSwapchainCreateInfoMETA  
 (type), [985](#)  
 XrEnvironmentDepthSwapchainMETA (type), [983](#)  
 XrEnvironmentDepthSwapchainStateMETA (type),  
[987](#)  
 xrEraseSpaceFB (function), [867](#)  
 XrEventDataBaseHeader (type), [52](#)  
 XrEventDataBuffer (type), [53](#)  
 XrEventDataDisplayRefreshRateChangedFB (type),  
[634](#)  
 XrEventDataEventsLost (type), [54](#)  
 XrEventDataEyeCalibrationChangedML (type),  
[1161](#)  
 XrEventDataHeadsetFitChangedML (type), [1159](#)  
 XrEventDataInstanceLossPending (type), [87](#)  
 XrEventDataInteractionProfileChanged (type), [269](#)  
 XrEventDataLocalizationChangedML (type), [1100](#)  
 XrEventDataMainSessionVisibilityChangedEXTX

(type), [1400](#)

XrEventDataMarkerTrackingUpdateVARJO (type), [1381](#)

XrEventDataPassthroughStateChangedFB (type), [764](#)

XrEventDataPerfSettingsEXT (type), [542](#)

XrEventDataReferenceSpaceChangePending (type), [150](#)

XrEventDataSceneCaptureCompleteFB (type), [813](#)

XrEventDataSessionStateChanged (type), [197](#)

XrEventDataSpaceEraseCompleteFB (type), [864](#)

XrEventDataSpaceListSaveCompleteFB (type), [870](#)

XrEventDataSpaceQueryCompleteFB (type), [850](#)

XrEventDataSpaceQueryResultsAvailableFB (type), [849](#)

XrEventDataSpaceSaveCompleteFB (type), [863](#)

XrEventDataSpaceSetStatusCompleteFB (type), [827](#)

XrEventDataSpaceShareCompleteFB (type), [856](#)

XrEventDataSpatialAnchorCreateCompleteFB (type), [826](#)

XrEventDataUserPresenceChangedEXT (type), [577](#)

XrEventDataVirtualKeyboardBackspaceMETA (type), [1077](#)

XrEventDataVirtualKeyboardCommitTextMETA (type), [1076](#)

XrEventDataVirtualKeyboardEnterMETA (type), [1078](#)

XrEventDataVirtualKeyboardHiddenMETA (type), [1080](#)

XrEventDataVirtualKeyboardShownMETA (type), [1079](#)

XrEventDataVisibilityMaskChangedKHR (type), [379](#)

XrEventDataViveTrackerConnectedHTCX (type), [1406](#)

XrExportedLocalizationMapML (type), [1116](#)

XrExtensionProperties (type), [79](#)

XrExtent2Df (type), [43](#)

XrExtent2Di (type), [44](#)

XrExtent3Df (type), [44](#)

XrExtent3DfEXT (type), [566](#)

XrExtent3DfFB (type), [798](#)

XrExtent3DfKHR (type), [1433](#)

XrExternalCameraAttachedToDeviceOCULUS (type), [1341](#)

XrExternalCameraExtrinsicsOCULUS (type), [1343](#)

XrExternalCameraIntrinsicsOCULUS (type), [1342](#)

XrExternalCameraOCULUS (type), [1344](#)

XrExternalCameraStatusFlagBitsOCULUS (type), [1340](#)

XrExternalCameraStatusFlagsOCULUS (type), [1340](#)

XrEyeCalibrationStatusML (type), [1161](#)

XrEyeExpressionHTC (type), [929](#)

XrEyeGazeFB (type), [648](#)

XrEyeGazeSampleTimeEXT (type), [470](#)

XrEyeGazesFB (type), [647](#)

XrEyeGazesInfoFB (type), [646](#)

XrEyePositionFB (type), [649](#)

XrEyeTrackerCreateInfoFB (type), [643](#)

XrEyeTrackerFB (type), [642](#)

XrEyeVisibility (type), [244](#)

## F

XR\_FACE\_EXPRESSION\_SET\_DEFAULT\_FB (define), [656](#)

XR\_FACIAL\_EXPRESSION\_EYE\_COUNT\_HTC (define), [929](#)

XR\_FACIAL\_EXPRESSION\_LIP\_COUNT\_HTC (define), [929](#)

XR\_FAILED (define), [30](#)

XR\_FALSE (define), [49](#)

XR\_FREQUENCY\_UNSPECIFIED (define), [286](#)

XrFaceConfidence2FB (type), [704](#)

XrFaceConfidenceFB (type), [665](#)

XrFaceExpression2FB (type), [680](#)

XrFaceExpressionFB (type), [663](#)

XrFaceExpressionInfo2FB (type), [675](#)

XrFaceExpressionInfoFB (type), [658](#)

XrFaceExpressionSet2FB (type), [672](#)

XrFaceExpressionSetFB (type), [656](#)

XrFaceExpressionStatusFB (type), [661](#)

XrFaceExpressionWeights2FB (type), [676](#)

XrFaceExpressionWeightsFB (type), [659](#)

XrFaceTracker2FB (type), [669](#)

XrFaceTrackerCreateInfo2FB (type), [670](#)

XrFaceTrackerCreateInfoFB (type), [655](#)

XrFaceTrackerFB (type), [654](#)

XrFaceTrackingDataSource2FB (type), [672](#)  
 XrFacialExpressionsHTC (type), [927](#)  
 XrFacialTrackerCreateInfoHTC (type), [924](#)  
 XrFacialTrackerHTC (type), [923](#)  
 XrFacialTrackingTypeHTC (type), [925](#)  
 XrFlags64 (type), [15](#)  
 XrForceFeedbackCurlApplyLocationMNDX (type), [1417](#)  
 XrForceFeedbackCurlApplyLocationsMNDX (type), [1416](#)  
 XrForceFeedbackCurlLocationMNDX (type), [1414](#)  
 XrFormFactor (type), [91](#)  
 XrFoveatedViewConfigurationViewVARJO (type), [1367](#)  
 XrFoveationApplyInfoHTC (type), [949](#)  
 XrFoveationConfigurationHTC (type), [954](#)  
 XrFoveationCustomModeInfoHTC (type), [953](#)  
 XrFoveationDynamicFB (type), [715](#)  
 XrFoveationDynamicFlagBitsHTC (type), [952](#)  
 XrFoveationDynamicFlagsHTC (type), [952](#)  
 XrFoveationDynamicModeInfoHTC (type), [951](#)  
     XrFoveationEyeTrackedProfileCreateFlagBit  
     sMETA (type), [999](#)  
 XrFoveationEyeTrackedProfileCreateFlagsMETA  
 (type), [999](#)  
 XrFoveationEyeTrackedProfileCreateInfoMETA  
 (type), [1000](#)  
 XrFoveationEyeTrackedStateFlagBitsMETA (type), [999](#)  
 XrFoveationEyeTrackedStateFlagsMETA (type), [999](#)  
 XrFoveationEyeTrackedStateMETA (type), [1000](#)  
 XrFoveationLevelFB (type), [715](#)  
 XrFoveationLevelHTC (type), [954](#)  
 XrFoveationLevelProfileCreateInfoFB (type), [716](#)  
 XrFoveationModeHTC (type), [950](#)  
 XrFoveationProfileCreateInfoFB (type), [709](#)  
 XrFoveationProfileFB (type), [707](#)  
 XrFovf (type), [48](#)  
 XrFrameBeginInfo (type), [231](#)  
 XrFrameEndInfo (type), [234](#)  
 XrFrameEndInfoFlagBitsML (type), [1094](#)  
 XrFrameEndInfoFlagsML (type), [1094](#)  
 XrFrameEndInfoML (type), [1095](#)  
 XrFrameState (type), [228](#)  
 XrFrameWaitInfo (type), [227](#)  
 XrFrustumf (type), [47](#)  
 XrFrustumfKHR (type), [1434](#)  
 XrFutureCancelInfoEXT (type), [482](#)  
 XrFutureCompletionBaseHeaderEXT (type), [478](#)  
 XrFutureCompletionEXT (type), [479](#)  
 XrFutureEXT (type), [474](#)  
 XrFuturePollInfoEXT (type), [476](#)  
 XrFuturePollResultEXT (type), [477](#)  
 XrFutureStateEXT (type), [482](#)

## G

XrGeometryInstanceCreateInfoFB (type), [758](#)  
 XrGeometryInstanceFB (type), [750](#)  
 xrGeometryInstanceSetTransformFB (function), [777](#)  
 XrGeometryInstanceTransformFB (type), [759](#)  
 xrGetActionStateBoolean (function), [273](#)  
 xrGetActionStateFloat (function), [275](#)  
 xrGetActionStatePose (function), [279](#)  
 xrGetActionStateVector2f (function), [277](#)  
 xrGetAudioInputDeviceGuidOculus (function), [1338](#)  
 xrGetAudioOutputDeviceGuidOculus (function), [1337](#)  
 xrGetBodySkeletonFB (function), [604](#)  
 xrGetControllerModelKeyMSFT (function), [1171](#)  
 xrGetControllerModelPropertiesMSFT (function), [1176](#)  
 xrGetControllerModelStateMSFT (function), [1179](#)  
 xrGetCurrentInteractionProfile (function), [266](#)  
 xrGetD3D11GraphicsRequirementsKHR  
 (function), [343](#)  
 xrGetD3D12GraphicsRequirementsKHR  
 (function), [350](#)  
 xrGetDeviceSampleRateFB (function), [738](#)  
 xrGetDisplayRefreshRateFB (function), [637](#)  
 xrGetEnvironmentDepthSwapchainStateMETA  
 (function), [986](#)  
 xrGetExportedLocalizationMapDataML (function), [1114](#)  
 xrGetEyeGazesFB (function), [645](#)



[xrGetFaceExpressionWeights2FB \(function\)](#), 673  
[xrGetFaceExpressionWeightsFB \(function\)](#), 657  
[xrGetFacialExpressionsHTC \(function\)](#), 926  
[xrGetFoveationEyeTrackedStateMETA \(function\)](#), 1002  
[xrGetHandMeshFB \(function\)](#), 731  
[xrGetInputSourceLocalizedName \(function\)](#), 293  
[xrGetInstanceProcAddr \(function\)](#), 56  
[xrGetInstanceProperties \(function\)](#), 85  
[xrGetMarkerDetectorStateML \(function\)](#), 1138  
[xrGetMarkerLengthML \(function\)](#), 1147  
[xrGetMarkerNumberML \(function\)](#), 1142  
[xrGetMarkerReprojectionErrorML \(function\)](#), 1146  
[xrGetMarkerSizeVARJO \(function\)](#), 1377  
[xrGetMarkersML \(function\)](#), 1140  
[xrGetMarkerStringML \(function\)](#), 1144  
[xrGetOpenGLGraphicsRequirementsKHR \(function\)](#), 373  
[xrGetOpenGLGraphicsRequirementsKHR \(function\)](#), 366  
[xrGetPassthroughPreferencesMETA \(function\)](#), 1023  
[xrGetPerformanceMetricsStateMETA \(function\)](#), 1032  
[xrGetPlaneDetectionsEXT \(function\)](#), 556  
[xrGetPlaneDetectionStateEXT \(function\)](#), 555  
[xrGetPlanePolygonBufferEXT \(function\)](#), 563  
[xrGetRecommendedLayerResolutionMETA \(function\)](#), 1037  
[xrGetReferenceSpaceBoundsRect \(function\)](#), 149  
[xrGetRenderModelPropertiesFB \(function\)](#), 792  
[xrGetSceneComponentsMSFT \(function\)](#), 1256  
[xrGetSceneComputeStateMSFT \(function\)](#), 1248  
[xrGetSceneMarkerDecodedStringMSFT \(function\)](#), 1226  
[xrGetSceneMarkerRawDataMSFT \(function\)](#), 1228  
[xrGetSceneMeshBuffersMSFT \(function\)](#), 1270  
[xrGetSerializedSceneFragmentDataMSFT \(function\)](#), 1285  
[xrGetSpaceBoundary2DFB \(function\)](#), 808  
[xrGetSpaceBoundingBox2DFB \(function\)](#), 804  
[xrGetSpaceBoundingBox3DFB \(function\)](#), 805  
[xrGetSpaceComponentStatusFB \(function\)](#), 834  
[xrGetSpaceContainerFB \(function\)](#), 839  
[xrGetSpaceRoomLayoutFB \(function\)](#), 809  
[xrGetSpaceSemanticLabelsFB \(function\)](#), 807  
[xrGetSpaceTriangleMeshMETA \(function\)](#), 1040  
[xrGetSpaceUserIdFB \(function\)](#), 877  
[xrGetSpaceUuidFB \(function\)](#), 830  
[xrGetSpatialAnchorNameHTC \(function\)](#), 919  
[xrGetSpatialGraphNodeBindingPropertiesMSFT \(function\)](#), 1331  
[xrGetSwapchainStateFB \(function\)](#), 882  
[xrGetSystem \(function\)](#), 92  
[xrGetSystemProperties \(function\)](#), 95  
[xrGetViewConfigurationProperties \(function\)](#), 178  
[xrGetVirtualKeyboardDirtyTexturesMETA \(function\)](#), 1063  
[xrGetVirtualKeyboardModelAnimationState sMETA \(function\)](#), 1067  
[xrGetVirtualKeyboardScaleMETA \(function\)](#), 1060  
[xrGetVirtualKeyboardTextureDataMETA \(function\)](#), 1065  
[xrGetVisibilityMaskKHR \(function\)](#), 380  
[xrGetVulkanDeviceExtensionsKHR \(function\)](#), 393  
[xrGetVulkanGraphicsDevice2KHR \(function\)](#), 402  
[xrGetVulkanGraphicsDeviceKHR \(function\)](#), 391  
[xrGetVulkanGraphicsRequirements2KHR \(function\)](#), 397  
[xrGetVulkanGraphicsRequirementsKHR \(function\)](#), 389  
[xrGetVulkanInstanceExtensionsKHR \(function\)](#), 392  
[XrGlobalDimmerFrameEndInfoFlagBitsML \(type\)](#), 1097  
[XrGlobalDimmerFrameEndInfoFlagsML \(type\)](#), 1097  
[XrGlobalDimmerFrameEndInfoML \(type\)](#), 1097  
[XrGraphicsBindingD3D11KHR \(type\)](#), 340  
[XrGraphicsBindingD3D12KHR \(type\)](#), 347  
[XrGraphicsBindingEGLMNDX \(type\)](#), 1412  
[XrGraphicsBindingOpenGLESAndroidKHR \(type\)](#), 370  
[XrGraphicsBindingOpenGLWaylandKHR \(type\)](#), 363  
[XrGraphicsBindingOpenGLWin32KHR \(type\)](#), 359

XrGraphicsBindingOpenGLXcbKHR (type), 362  
XrGraphicsBindingOpenGLXlibKHR (type), 360  
XrGraphicsBindingVulkan2KHR (type), 408  
XrGraphicsBindingVulkanKHR (type), 386  
XrGraphicsRequirementsD3D11KHR (type), 342  
XrGraphicsRequirementsD3D12KHR (type), 349  
XrGraphicsRequirementsOpenGLESKHR (type),  
372  
XrGraphicsRequirementsOpenGLKHR (type), 365  
XrGraphicsRequirementsVulkan2KHR (type), 398  
XrGraphicsRequirementsVulkanKHR (type), 388

## H

XR\_HAND\_FOREARM\_JOINT\_COUNT\_ULTRALEAP  
(define), 1355  
XR\_HAND\_JOINT\_COUNT\_EXT (define), 526  
XrHandCapsuleFB (type), 724  
XrHandEXT (type), 513  
XrHandForearmJointULTRALEAP (type), 1354  
XrHandJointEXT (type), 524  
XrHandJointLocationEXT (type), 518  
XrHandJointLocationsEXT (type), 517  
XrHandJointSetEXT (type), 513  
XrHandJointsLocateInfoEXT (type), 516  
XrHandJointsMotionRangeEXT (type), 507  
XrHandJointsMotionRangeInfoEXT (type), 508  
XrHandJointVelocitiesEXT (type), 519  
XrHandJointVelocityEXT (type), 521  
XrHandMeshIndexBufferMSFT (type), 1198  
XrHandMeshMSFT (type), 1196  
XrHandMeshSpaceCreateInfoMSFT (type), 1192  
XrHandMeshUpdateInfoMSFT (type), 1195  
XrHandMeshVertexBufferMSFT (type), 1199  
XrHandMeshVertexMSFT (type), 1201  
XrHandPoseTypeInfoMSFT (type), 1204  
XrHandPoseTypeMSFT (type), 1204  
XrHandTrackerCreateInfoEXT (type), 512  
XrHandTrackerEXT (type), 510  
XrHandTrackingAimFlagBitsFB (type), 720  
XrHandTrackingAimFlagsFB (type), 720  
XrHandTrackingAimStateFB (type), 722  
XrHandTrackingCapsulesStateFB (type), 725  
XrHandTrackingDataSourceEXT (type), 529  
XrHandTrackingDataSourceInfoEXT (type), 530

XrHandTrackingDataSourceStateEXT (type), 531  
XrHandTrackingMeshFB (type), 728  
XrHandTrackingScaleFB (type), 730  
XrHapticActionInfo (type), 272  
XrHapticAmplitudeEnvelopeVibrationFB (type),  
734  
XrHapticBaseHeader (type), 284  
XrHapticPcmVibrationFB (type), 736  
XrHapticVibration (type), 285  
XrHeadsetFitStatusML (type), 1160  
XrHolographicWindowAttachmentMSFT (type),  
1209

## I

XR\_INFINITE\_DURATION (define), 37  
xrImportLocalizationMapML (function), 1109  
xrInitializeLoaderKHR (function), 354  
XrInputSourceLocalizedNameFlagBits (type), 295  
XrInputSourceLocalizedNameFlags (type), 295  
XrInputSourceLocalizedNameGetInfo (type), 294  
XrInstance (type), 74  
XrInstanceCreateFlagBits (type), 83  
XrInstanceCreateFlags (type), 82  
XrInstanceCreateInfo (type), 81  
XrInstanceCreateInfoAndroidKHR (type), 303  
XrInstanceProperties (type), 86  
XrInteractionProfileAnalogThresholdVALVE  
(type), 1357  
XrInteractionProfileDpadBindingEXT (type), 462  
XrInteractionProfileState (type), 268  
XrInteractionProfileSuggestedBinding (type), 263

## K

XrKeyboardSpaceCreateInfoFB (type), 745  
XrKeyboardTrackingDescriptionFB (type), 745  
XrKeyboardTrackingFlagBitsFB (type), 742  
XrKeyboardTrackingFlagsFB (type), 741  
XrKeyboardTrackingQueryFB (type), 744  
XrKeyboardTrackingQueryFlagBitsFB (type), 742  
XrKeyboardTrackingQueryFlagsFB (type), 742

## L

XR\_LOADER\_INFO\_STRUCT\_VERSION (define), 63  
XrLipExpressionHTC (type), 934  
xrLoadControllerModelMSFT (function), 1174

XrLoaderInitInfoAndroidKHR (type), 356  
 XrLoaderInitInfoBaseHeaderKHR (type), 353  
 XrLoaderInterfaceStructs (type), 63  
 xrLoadRenderModelFB (function), 793  
 XrLocalDimmingFrameEndInfoMETA (type), 1008  
 XrLocalDimmingModeMETA (type), 1008  
 XrLocalizationEnableEventsInfoML (type), 1103  
 XrLocalizationMapConfidenceML (type), 1117  
 XrLocalizationMapErrorFlagBitsML (type), 1101  
 XrLocalizationMapErrorFlagsML (type), 1101  
 XrLocalizationMapImportInfoML (type), 1111  
 XrLocalizationMapML (type), 1104  
 XrLocalizationMapQueryInfoBaseHeaderML (type), 1106  
 XrLocalizationMapStateML (type), 1117  
 XrLocalizationMapTypeML (type), 1118  
 xrLocateBodyJointsFB (function), 600  
 xrLocateHandJointsEXT (function), 515  
 xrLocateSceneComponentsMSFT (function), 1276  
 xrLocateSpace (function), 159  
 xrLocateSpaces (function), 166  
 xrLocateSpacesKHR (function), 1423  
 xrLocateViews (function), 220

**M**

XR\_MAKE\_VERSION (define), 1492  
 XR\_MAX\_EVENT\_DATA\_SIZE (define), 54

XR\_MAX\_EXTERNAL\_CAMERA\_NAME\_SIZE\_OCULUS (define), 1341

XR\_MAX\_HAPTIC\_AMPLITUDE\_ENVELOPE\_SAMPLES\_FB (define), 735

XR\_MAX\_HAPTIC\_PCM\_BUFFER\_SIZE\_FB (define), 740

XR\_MAY\_ALIAS (define), 13

XR\_MIN\_COMPOSITION\_LAYERS\_SUPPORTED (define), 98

XR\_MIN\_HAPTIC\_DURATION (define), 286

XrMapLocalizationRequestInfoML (type), 1108  
 XrMarkerAprilTagDictML (type), 1128  
 XrMarkerArucoDictML (type), 1125  
 XrMarkerDetectorAprilTagInfoML (type), 1127  
 XrMarkerDetectorArucoInfoML (type), 1124

XrMarkerDetectorCameraML (type), 1134  
 XrMarkerDetectorCornerRefineMethodML (type), 1134  
 XrMarkerDetectorCreateInfoML (type), 1121  
 XrMarkerDetectorCustomProfileInfoML (type), 1131  
 XrMarkerDetectorFpsML (type), 1133  
 XrMarkerDetectorFullAnalysisIntervalML (type), 1135  
 XrMarkerDetectorML (type), 1119  
 XrMarkerDetectorProfileML (type), 1122  
 XrMarkerDetectorResolutionML (type), 1133  
 XrMarkerDetectorSizeInfoML (type), 1129  
 XrMarkerDetectorSnapshotInfoML (type), 1137  
 XrMarkerDetectorStateML (type), 1139  
 XrMarkerDetectorStatusML (type), 1140  
 XrMarkerML (type), 1142  
 XrMarkerSpaceCreateInfoML (type), 1151  
 XrMarkerSpaceCreateInfoVARJO (type), 1382  
 XrMarkerTypeML (type), 1123  
 XrMeshComputeLodMSFT (type), 1246

**N**

XR\_NO\_DURATION (define), 37  
 XR\_NULL\_CONTROLLER\_MODEL\_KEY\_MSFT (define), 1173  
 XR\_NULL\_HANDLE (define), 1494  
 XR\_NULL\_PATH (define), 101  
 XR\_NULL\_RENDER\_MODEL\_KEY\_FB (define), 784  
 XR\_NULL\_SYSTEM\_ID (define), 92  
 XrNegotiateApiLayerRequest (type), 67  
 xrNegotiateLoaderApiLayerInterface (function), 65  
 XrNegotiateLoaderInfo (type), 62  
 xrNegotiateLoaderRuntimeInterface (function), 60  
 XrNegotiateRuntimeRequest (type), 64  
 XrNewSceneComputeInfoMSFT (type), 1239

**O**

XrObjectType (type), 32  
 XrOffset2Df (type), 42  
 XrOffset2Di (type), 43  
 XrOffset3DfFB (type), 798  
 XrOverlayMainSessionFlagBitsEXTX (type), 1399  
 XrOverlayMainSessionFlagsEXTX (type), 1399

XrOverlaySessionCreateFlagBitsEXTX (type), [1399](#)  
XrOverlaySessionCreateFlagsEXTX (type), [1399](#)

## P

PFN\_xrCreateApiLayerInstance, [59](#)  
PFN\_xrDebugUtilsMessengerCallbackEXT, [451](#)  
PFN\_xrEglGetProcAddressMNDX, [1413](#)  
PFN\_xrGetInstanceProcAddr, [59](#)  
PFN\_xrVoidFunction, [59](#)  
XrPassthroughBrightnessContrastSaturationFB (type), [763](#)  
XrPassthroughCapabilityFlagBitsFB (type), [752](#)  
XrPassthroughCapabilityFlagsFB (type), [751](#)  
XrPassthroughColorHTC (type), [964](#)  
XrPassthroughColorLutChannelsMETA (type), [1011](#)  
XrPassthroughColorLutCreateInfoMETA (type), [1012](#)  
XrPassthroughColorLutDataMETA (type), [1012](#)  
XrPassthroughColorLutMETA (type), [1010](#)  
XrPassthroughColorLutUpdateInfoMETA (type), [1014](#)  
XrPassthroughColorMapInterpolatedLutMETA (type), [1016](#)  
XrPassthroughColorMapLutMETA (type), [1015](#)  
XrPassthroughColorMapMonoToMonoFB (type), [762](#)  
XrPassthroughColorMapMonoToRgbaFB (type), [761](#)  
XrPassthroughCreateInfoFB (type), [755](#)  
XrPassthroughCreateInfoHTC (type), [960](#)  
XrPassthroughFB (type), [750](#)  
XrPassthroughFlagBitsFB (type), [750](#)  
XrPassthroughFlagsFB (type), [750](#)  
XrPassthroughFormHTC (type), [961](#)  
XrPassthroughHTC (type), [959](#)  
XrPassthroughKeyboardHandsIntensityFB (type), [780](#)  
XrPassthroughLayerCreateInfoFB (type), [756](#)  
XrPassthroughLayerFB (type), [750](#)  
xrPassthroughLayerPauseFB (function), [771](#)  
XrPassthroughLayerPurposeFB (type), [753](#)  
xrPassthroughLayerResumeFB (function), [772](#)

xrPassthroughLayerSetKeyboardHandsIntensityFB (function), [781](#)  
xrPassthroughLayerSetStyleFB (function), [773](#)  
XrPassthroughMeshTransformInfoHTC (type), [965](#)  
xrPassthroughPauseFB (function), [768](#)  
XrPassthroughPreferenceFlagBitsMETA (type), [1022](#)  
XrPassthroughPreferenceFlagsMETA (type), [1022](#)  
XrPassthroughPreferencesMETA (type), [1022](#)  
xrPassthroughStartFB (function), [767](#)  
XrPassthroughStateChangedFlagBitsFB (type), [751](#)  
XrPassthroughStateChangedFlagsFB (type), [751](#)  
XrPassthroughStyleFB (type), [760](#)  
XrPath (type), [99](#)  
xrPathToString (function), [103](#)  
XrPerformanceMetricsCounterFlagBitsMETA (type), [1026](#)  
XrPerformanceMetricsCounterFlagsMETA (type), [1026](#)  
XrPerformanceMetricsCounterMETA (type), [1028](#)  
XrPerformanceMetricsCounterUnitMETA (type), [1027](#)  
XrPerformanceMetricsStateMETA (type), [1028](#)  
XrPerfSettingsDomainEXT (type), [534](#)  
XrPerfSettingsLevelEXT (type), [535](#)  
XrPerfSettingsNotificationLevelEXT (type), [543](#)  
xrPerfSettingsSetPerformanceLevelEXT (function), [541](#)  
XrPerfSettingsSubDomainEXT (type), [543](#)  
xrPersistSpatialAnchorMSFT (function), [1314](#)  
XrPlaneDetectionCapabilityFlagBitsEXT (type), [546](#)  
XrPlaneDetectionCapabilityFlagsEXT (type), [546](#)  
XrPlaneDetectionStateEXT (type), [563](#)  
XrPlaneDetectorBeginInfoEXT (type), [553](#)  
XrPlaneDetectorCreateInfoEXT (type), [549](#)  
XrPlaneDetectorEXT (type), [547](#)  
XrPlaneDetectorFlagBitsEXT (type), [550](#)  
XrPlaneDetectorFlagsEXT (type), [550](#)  
XrPlaneDetectorGetInfoEXT (type), [558](#)  
XrPlaneDetectorLocationEXT (type), [560](#)  
XrPlaneDetectorLocationsEXT (type), [559](#)  
XrPlaneDetectorOrientationEXT (type), [561](#)  
XrPlaneDetectorPolygonBufferEXT (type), [565](#)  
XrPlaneDetectorSemanticTypeEXT (type), [562](#)

[xrPollEvent \(function\)](#), [50](#)  
[xrPollFutureEXT \(function\)](#), [475](#)  
[XrPosef \(type\)](#), [41](#)

**Q**

[XrQuaternionf \(type\)](#), [41](#)  
[xrQueryLocalizationMapsML \(function\)](#), [1105](#)  
[xrQueryPerformanceMetricsCounterMETA \(function\)](#), [1033](#)  
[xrQuerySpacesFB \(function\)](#), [851](#)  
[xrQuerySystemTrackedKeyboardFB \(function\)](#), [746](#)

**R**

[XR\\_RUNTIME\\_INFO\\_STRUCT\\_VERSION \(define\)](#), [65](#)  
[XrRecommendedLayerResolutionGetInfoMETA \(type\)](#), [1036](#)  
[XrRecommendedLayerResolutionMETA \(type\)](#), [1035](#)  
[XrRect2Df \(type\)](#), [45](#)  
[XrRect2Di \(type\)](#), [46](#)  
[XrRect3DfFB \(type\)](#), [799](#)  
[XrReferenceSpaceCreateInfo \(type\)](#), [155](#)  
[XrReferenceSpaceType \(type\)](#), [145](#)  
[xrReleaseSwapchainImage \(function\)](#), [218](#)  
[XrRenderModelBufferFB \(type\)](#), [789](#)  
[XrRenderModelCapabilitiesRequestFB \(type\)](#), [787](#)  
[XrRenderModelFlagBitsFB \(type\)](#), [783](#)  
[XrRenderModelFlagsFB \(type\)](#), [783](#)  
[XrRenderModelKeyFB \(type\)](#), [784](#)  
[XrRenderModelLoadInfoFB \(type\)](#), [788](#)  
[XrRenderModelPathInfoFB \(type\)](#), [785](#)  
[XrRenderModelPropertiesFB \(type\)](#), [786](#)  
[XrReprojectionModeMSFT \(type\)](#), [1167](#)  
[xrRequestDisplayRefreshRateFB \(function\)](#), [638](#)  
[xrRequestExitSession \(function\)](#), [196](#)  
[xrRequestMapLocalizationML \(function\)](#), [1107](#)  
[xrRequestSceneCaptureFB \(function\)](#), [814](#)  
[XrResult \(type\)](#), [19](#)  
[xrResultToString \(function\)](#), [88](#)  
[xrRetrieveSpaceQueryResultsFB \(function\)](#), [852](#)  
[XrRoomLayoutFB \(type\)](#), [801](#)

**S**

[XR\\_SUCCEEDED \(define\)](#), [30](#)  
[xrSaveSpaceFB \(function\)](#), [865](#)  
[xrSaveSpaceListFB \(function\)](#), [872](#)  
[XrSceneBoundsMSFT \(type\)](#), [1242](#)  
[XrSceneCaptureRequestInfoFB \(type\)](#), [812](#)  
[XrSceneComponentLocationMSFT \(type\)](#), [1280](#)  
[XrSceneComponentLocationsMSFT \(type\)](#), [1279](#)  
[XrSceneComponentMSFT \(type\)](#), [1259](#)  
[XrSceneComponentParentFilterInfoMSFT \(type\)](#), [1260](#)  
[XrSceneComponentsGetInfoMSFT \(type\)](#), [1257](#)  
[XrSceneComponentsLocateInfoMSFT \(type\)](#), [1278](#)  
[XrSceneComponentsMSFT \(type\)](#), [1258](#)  
[XrSceneComponentTypeMSFT \(type\)](#), [1255](#)  
[XrSceneComputeConsistencyMSFT \(type\)](#), [1241](#)  
[XrSceneComputeFeatureMSFT \(type\)](#), [1240](#)  
[XrSceneComputeStateMSFT \(type\)](#), [1249](#)  
[XrSceneCreateInfoMSFT \(type\)](#), [1252](#)  
[XrSceneDeserializeInfoMSFT \(type\)](#), [1289](#)  
[XrSceneFrustumBoundMSFT \(type\)](#), [1245](#)  
[XrSceneMarkerMSFT \(type\)](#), [1219](#)  
[XrSceneMarkerQRCodeMSFT \(type\)](#), [1225](#)  
[XrSceneMarkerQRCodesMSFT \(type\)](#), [1224](#)  
[XrSceneMarkerQRCodeSymbolTypeMSFT \(type\)](#), [1226](#)  
[XrSceneMarkersMSFT \(type\)](#), [1218](#)  
[XrSceneMarkerTypeFilterMSFT \(type\)](#), [1220](#)  
[XrSceneMarkerTypeMSFT \(type\)](#), [1221](#)  
[XrSceneMeshBuffersGetInfoMSFT \(type\)](#), [1272](#)  
[XrSceneMeshBuffersMSFT \(type\)](#), [1273](#)  
[XrSceneMeshesMSFT \(type\)](#), [1269](#)  
[XrSceneMeshIndicesUint16MSFT \(type\)](#), [1275](#)  
[XrSceneMeshIndicesUint32MSFT \(type\)](#), [1274](#)  
[XrSceneMeshMSFT \(type\)](#), [1269](#)  
[XrSceneMeshVertexBufferMSFT \(type\)](#), [1273](#)  
[XrSceneMSFT \(type\)](#), [1239](#)  
[XrSceneObjectMSFT \(type\)](#), [1264](#)  
[XrSceneObjectsMSFT \(type\)](#), [1263](#)  
[XrSceneObjectTypeMSFT \(type\)](#), [1265](#)  
[XrSceneObjectTypesFilterInfoMSFT \(type\)](#), [1261](#)  
[XrSceneObserverCreateInfoMSFT \(type\)](#), [1234](#)  
[XrSceneObserverMSFT \(type\)](#), [1233](#)  
[XrSceneOrientedBoxBoundMSFT \(type\)](#), [1244](#)  
[XrScenePlaneAlignmentFilterInfoMSFT \(type\)](#), [1262](#)

[XrScenePlaneAlignmentTypeMSFT \(type\), 1268](#)  
[XrScenePlaneMSFT \(type\), 1266](#)  
[XrScenePlanesMSFT \(type\), 1266](#)  
[XrSceneSphereBoundMSFT \(type\), 1243](#)  
  
[XrSecondaryViewConfigurationFrameEndInfoMSFT \(type\), 1299](#)  
[XrSecondaryViewConfigurationFrameStateMSFT \(type\), 1296](#)  
[XrSecondaryViewConfigurationLayerInfoMSFT \(type\), 1300](#)  
  
[XrSecondaryViewConfigurationSessionBeginInfoMSFT \(type\), 1295](#)  
[XrSecondaryViewConfigurationStateMSFT \(type\), 1297](#)  
  
[XrSecondaryViewConfigurationSwapchainCreateInfoMSFT \(type\), 1293](#)  
[XrSemanticLabelsFB \(type\), 800](#)  
[XrSemanticLabelsSupportFlagBitsFB \(type\), 796](#)  
[XrSemanticLabelsSupportFlagsFB \(type\), 796](#)  
[XrSemanticLabelsSupportInfoFB \(type\), 803](#)  
[xrSendVirtualKeyboardInputMETA \(function\), 1071](#)  
[XrSerializedSceneFragmentDataGetInfoMSFT \(type\), 1286](#)  
[XrSession \(type\), 186](#)  
[XrSessionActionSetsAttachInfo \(type\), 266](#)  
[xrSessionBeginDebugUtilsLabelRegionEXT \(function\), 447](#)  
[XrSessionBeginInfo \(type\), 194](#)  
[XrSessionCreateFlagBits \(type\), 191](#)  
[XrSessionCreateFlags \(type\), 191](#)  
[XrSessionCreateInfo \(type\), 190](#)  
[XrSessionCreateInfoOverlayEXTX \(type\), 1399](#)  
[xrSessionEndDebugUtilsLabelRegionEXT \(function\), 448](#)  
[xrSessionInsertDebugUtilsLabelEXT \(function\), 449](#)  
[XrSessionState \(type\), 198](#)  
[xrSetAndroidApplicationThreadKHR \(function\), 309](#)  
[xrSetColorSpaceFB \(function\), 617](#)  
  
[xrSetDebugUtilsObjectNameEXT \(function\), 442](#)  
[xrSetDigitalLensControlALMALENCE \(function\), 588](#)  
[xrSetEnvironmentDepthEstimationVARJO \(function\), 1364](#)  
[xrSetEnvironmentDepthHandRemovalMETA \(function\), 981](#)  
[xrSetInputDeviceActiveEXT \(function\), 424](#)  
[xrSetInputDeviceLocationEXT \(function\), 429](#)  
[xrSetInputDeviceStateBoolEXT \(function\), 425](#)  
[xrSetInputDeviceStateFloatEXT \(function\), 427](#)  
[xrSetInputDeviceStateVector2fEXT \(function\), 428](#)  
[xrSetMarkerTrackingPredictionVARJO \(function\), 1376](#)  
[xrSetMarkerTrackingTimeoutVARJO \(function\), 1375](#)  
[xrSetMarkerTrackingVARJO \(function\), 1374](#)  
[xrSetPerformanceMetricsStateMETA \(function\), 1031](#)  
[xrSetSpaceComponentStatusFB \(function\), 833](#)  
[xrSetTrackingOptimizationSettingsHintQCOM \(function\), 1351](#)  
[xrSetViewOffsetVARJO \(function\), 1387](#)  
[xrSetVirtualKeyboardModelVisibilityMETA \(function\), 1061](#)  
[xrShareSpacesFB \(function\), 858](#)  
[xrSnapshotMarkerDetectorML \(function\), 1136](#)  
[XrSpace \(type\), 144](#)  
[XrSpaceComponentFilterInfoFB \(type\), 847](#)  
[XrSpaceComponentStatusFB \(type\), 825](#)  
[XrSpaceComponentStatusSetInfoFB \(type\), 824](#)  
[XrSpaceComponentTypeFB \(type\), 822](#)  
[XrSpaceContainerFB \(type\), 837](#)  
[XrSpaceEraseInfoFB \(type\), 862](#)  
[XrSpaceFilterInfoBaseHeaderFB \(type\), 843](#)  
[XrSpaceListSaveInfoFB \(type\), 869](#)  
[XrSpaceLocation \(type\), 162](#)  
[XrSpaceLocationData \(type\), 169](#)  
[XrSpaceLocationDataKHR \(type\), 1427](#)  
[XrSpaceLocationFlagBits \(type\), 163](#)  
[XrSpaceLocationFlags \(type\), 163](#)  
[XrSpaceLocations \(type\), 168](#)  
[XrSpaceLocationsKHR \(type\), 1425](#)  
[XrSpacePersistenceModeFB \(type\), 861](#)

XrSpaceQueryActionFB (type), [841](#)  
 XrSpaceQueryInfoBaseHeaderFB (type), [842](#)  
 XrSpaceQueryInfoFB (type), [844](#)  
 XrSpaceQueryResultFB (type), [848](#)  
 XrSpaceQueryResultsFB (type), [848](#)  
 XrSpaceSaveInfoFB (type), [862](#)  
 XrSpaceShareInfoFB (type), [855](#)  
 XrSpacesLocateInfo (type), [167](#)  
 XrSpacesLocateInfoKHR (type), [1424](#)  
 XrSpaceStorageLocationFB (type), [861](#)  
 XrSpaceStorageLocationFilterInfoFB (type), [845](#)  
 XrSpaceTriangleMeshGetInfoMETA (type), [1041](#)  
 XrSpaceTriangleMeshMETA (type), [1042](#)  
 XrSpaceUserCreateInfoFB (type), [875](#)  
 XrSpaceUserFB (type), [874](#)  
 XrSpaceUserIdFB (type), [875](#)  
 XrSpaceUuidFilterInfoFB (type), [846](#)  
 XrSpaceVelocities (type), [170](#)  
 XrSpaceVelocitiesKHR (type), [1427](#)  
 XrSpaceVelocity (type), [164](#)  
 XrSpaceVelocityData (type), [171](#)  
 XrSpaceVelocityDataKHR (type), [1428](#)  
 XrSpaceVelocityFlagBits (type), [165](#)  
 XrSpaceVelocityFlags (type), [165](#)  
 XrSpatialAnchorCreateInfoFB (type), [823](#)  
 XrSpatialAnchorCreateInfoHTC (type), [918](#)  
 XrSpatialAnchorCreateInfoMSFT (type), [1305](#)  
  
     XrSpatialAnchorFromPersistedAnchorCreateInfoMSFT (type), [1319](#)  
 XrSpatialAnchorMSFT (type), [1303](#)  
 XrSpatialAnchorNameHTC (type), [918](#)  
 XrSpatialAnchorPersistenceInfoMSFT (type), [1315](#)  
 XrSpatialAnchorPersistenceNameMSFT (type), [1315](#)  
 XrSpatialAnchorSpaceCreateInfoMSFT (type), [1307](#)  
 XrSpatialAnchorStoreConnectionMSFT (type), [1311](#)  
 XrSpatialGraphNodeBindingMSFT (type), [1327](#)  
  
     XrSpatialGraphNodeBindingPropertiesGetInfoMSFT (type), [1332](#)  
 XrSpatialGraphNodeBindingPropertiesMSFT (type), [1333](#)  
 XrSpatialGraphNodeSpaceCreateInfoMSFT (type), [1325](#)  
 XrSpatialGraphNodeTypeMSFT (type), [1326](#)  
 XrSpatialGraphStaticNodeBindingCreateInfoMSFT (type), [1329](#)  
 XrSpheref (type), [46](#)  
 XrSpherefKHR (type), [1433](#)  
 xrStartEnvironmentDepthProviderMETA (function), [978](#)  
 xrStopEnvironmentDepthProviderMETA (function), [980](#)  
 xrStopHapticFeedback (function), [286](#)  
 xrStringToPath (function), [101](#)  
 XrStructureType (type), [1477](#)  
 xrStructureTypeToString (function), [89](#)  
 xrSubmitDebugUtilsMessageEXT (function), [446](#)  
 xrSuggestInteractionProfileBindings (function), [261](#)  
 xrSuggestVirtualKeyboardLocationMETA (function), [1057](#)  
 XrSwapchain (type), [202](#)  
 XrSwapchainCreateFlagBits (type), [208](#)  
 XrSwapchainCreateFlags (type), [208](#)  
 XrSwapchainCreateFoveationFlagBitsFB (type), [707](#)  
 XrSwapchainCreateFoveationFlagsFB (type), [707](#)  
 XrSwapchainCreateInfo (type), [205](#)  
 XrSwapchainCreateInfoFoveationFB (type), [709](#)  
 XrSwapchainImageAcquireInfo (type), [215](#)  
 XrSwapchainImageBaseHeader (type), [212](#)  
 XrSwapchainImageD3D11KHR (type), [341](#)  
 XrSwapchainImageD3D12KHR (type), [348](#)  
 XrSwapchainImageFoveationVulkanFB (type), [718](#)  
 XrSwapchainImageOpenGLESKHR (type), [371](#)  
 XrSwapchainImageOpenGLKHR (type), [364](#)  
 XrSwapchainImageReleaseInfo (type), [219](#)  
 XrSwapchainImageVulkan2KHR (type), [410](#)  
 XrSwapchainImageVulkanKHR (type), [387](#)  
 XrSwapchainImageWaitInfo (type), [217](#)  
 XrSwapchainStateAndroidSurfaceDimensionsFB (type), [885](#)  
 XrSwapchainStateBaseHeaderFB (type), [880](#)  
 XrSwapchainStateFoveationFB (type), [710](#)

XrSwapchainStateFoveationFlagBitsFB (type), 708  
 XrSwapchainStateFoveationFlagsFB (type), 708  
 XrSwapchainStateSamplerOpenGLESB (type), 887  
 XrSwapchainStateSamplerVulkanFB (type), 890  
 XrSwapchainSubImage (type), 239  
 XrSwapchainUsageFlagBits (type), 209  
 XrSwapchainUsageFlags (type), 208  
 xrSyncActions (function), 287  
 XrSystemAnchorPropertiesHTC (type), 915  
 XrSystemBodyTrackingPropertiesFB (type), 595  
 XrSystemColorSpacePropertiesFB (type), 615  
 XrSystemEnvironmentDepthPropertiesMETA (type), 974  
 XrSystemEyeGazeInteractionPropertiesEXT (type), 468  
 XrSystemEyeTrackingPropertiesFB (type), 641  
 XrSystemFaceTrackingProperties2FB (type), 668  
 XrSystemFaceTrackingPropertiesFB (type), 653  
 XrSystemFacialTrackingPropertiesHTC (type), 922  
 XrSystemForceFeedbackCurlPropertiesMNDX (type), 1415  
 XrSystemFoveatedRenderingPropertiesVARJO (type), 1367  
 XrSystemFoveationEyeTrackedPropertiesMETA (type), 1001  
 XrSystemGetInfo (type), 93  
 XrSystemGraphicsProperties (type), 97  
 XrSystemHandTrackingMeshPropertiesMSFT (type), 1189  
 XrSystemHandTrackingPropertiesEXT (type), 509  
 XrSystemHeadsetIdPropertiesMETA (type), 1005  
 XrSystemId (type), 92  
 XrSystemKeyboardTrackingPropertiesFB (type), 743  
 XrSystemMarkerTrackingPropertiesVARJO (type), 1380  
 XrSystemMarkerUnderstandingPropertiesML (type), 1156  
 XrSystemPassthroughColorLutPropertiesMETA (type), 1011  
 XrSystemPassthroughProperties2FB (type), 754  
 XrSystemPassthroughPropertiesFB (type), 754  
 XrSystemPlaneDetectionPropertiesEXT (type), 545  
 XrSystemProperties (type), 96  
 XrSystemRenderModelPropertiesFB (type), 784  
 XrSystemSpaceWarpPropertiesFB (type), 819  
 XrSystemSpatialEntityPropertiesFB (type), 823  
 XrSystemTrackingProperties (type), 98  
 XrSystemUserPresencePropertiesEXT (type), 576  
 XrSystemVirtualKeyboardPropertiesMETA (type), 1050

## T

XR\_TRUE (define), 49  
 xrThermalGetTemperatureTrendEXT (function), 573  
 XrTime (type), 36  
 XrTrackingOptimizationSettingsDomainQCOM (type), 1350  
 XrTrackingOptimizationSettingsHintQCOM (type), 1350  
 xrTriangleMeshBeginUpdateFB (function), 909  
 xrTriangleMeshBeginVertexBufferUpdateFB (function), 912  
 XrTriangleMeshCreateInfoFB (type), 902  
 xrTriangleMeshEndUpdateFB (function), 910  
 xrTriangleMeshEndVertexBufferUpdateFB (function), 913  
 XrTriangleMeshFB (type), 900  
 XrTriangleMeshFlagBitsFB (type), 900  
 XrTriangleMeshFlagsFB (type), 900  
 xrTriangleMeshGetIndexBufferFB (function), 908  
 xrTriangleMeshGetVertexBufferFB (function), 906  
 xrTryCreateSpatialGraphStaticNodeBindingMSFT (function), 1327  
 xrTryGetPerceptionAnchorFromSpatialAnchorMSFT (function), 1214

## U

XR\_UNQUALIFIED\_SUCCESS (define), 31  
 xrUnpersistSpatialAnchorMSFT (function), 1320  
 xrUpdateHandMeshMSFT (function), 1193  
 xrUpdatePassthroughColorLutMETA (function), 1019  
 xrUpdateSwapchainFB (function), 881  
 XrUserCalibrationEnableEventsInfoML (type), 1158



XrUuid (type), [48](#)  
XrUuidEXT (type), [1447](#)  
XrUuidMSFT (type), [1254](#)

## V

XR\_VERSION\_MAJOR (define), [1493](#)  
XR\_VERSION\_MINOR (define), [1493](#)  
XR\_VERSION\_PATCH (define), [1493](#)  
XrVector2f (type), [40](#)  
XrVector3f (type), [40](#)  
XrVector4f (type), [40](#)  
XrVector4sFB (type), [727](#)  
XrVersion (type), [6](#)  
XrView (type), [222](#)  
XrViewConfigurationDepthRangeEXT (type), [583](#)  
XrViewConfigurationProperties (type), [179](#)  
XrViewConfigurationType (type), [174](#)  
XrViewConfigurationView (type), [182](#)  
XrViewConfigurationViewFovEPIC (type), [590](#)  
XrViewLocateFoveatedRenderingVARJO (type),  
[1371](#)  
XrViewLocateInfo (type), [221](#)  
XrViewState (type), [223](#)  
XrViewStateFlagBits (type), [224](#)  
XrViewStateFlags (type), [224](#)  
XrVirtualKeyboardAnimationStateMETA (type),  
[1069](#)  
XrVirtualKeyboardCreateInfoMETA (type), [1053](#)  
XrVirtualKeyboardInputInfoMETA (type), [1073](#)  
XrVirtualKeyboardInputSourceMETA (type), [1086](#)  
XrVirtualKeyboardInputStateFlagBitsMETA (type),  
[1084](#)  
XrVirtualKeyboardInputStateFlagsMETA (type),  
[1084](#)  
XrVirtualKeyboardLocationInfoMETA (type), [1058](#)  
XrVirtualKeyboardLocationTypeMETA (type),  
[1085](#)  
XrVirtualKeyboardMETA (type), [1084](#)  
XrVirtualKeyboardModelAnimationStatesMETA  
(type), [1070](#)  
XrVirtualKeyboardModelVisibilitySetInfoMETA  
(type), [1062](#)  
XrVirtualKeyboardSpaceCreateInfoMETA (type),  
[1056](#)

XrVirtualKeyboardTextContextChangeInfoMETA  
(type), [1075](#)  
XrVirtualKeyboardTextureDataMETA (type), [1066](#)  
XrVisibilityMaskKHR (type), [378](#)  
XrVisibilityMaskTypeKHR (type), [377](#)  
XrVisualMeshComputeLodInfoMSFT (type), [1246](#)  
XrViveTrackerPathsHTCX (type), [1405](#)  
XrVulkanDeviceCreateFlagBitsKHR (type), [407](#)  
XrVulkanDeviceCreateFlagsKHR (type), [407](#)  
XrVulkanDeviceCreateInfoKHR (type), [405](#)  
XrVulkanGraphicsDeviceGetInfoKHR (type), [403](#)  
XrVulkanInstanceCreateFlagBitsKHR (type), [402](#)  
XrVulkanInstanceCreateFlagsKHR (type), [402](#)  
XrVulkanInstanceCreateInfoKHR (type), [400](#)  
XrVulkanSwapchainCreateInfoMETA (type), [1089](#)  
XrVulkanSwapchainFormatListCreateInfoKHR  
(type), [414](#)

## W

xrWaitFrame (function), [225](#)  
xrWaitSwapchainImage (function), [216](#)  
XrWindingOrderFB (type), [901](#)