**OpenXR** is a cross-platform API that enables 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, augmented reality, and mixed reality. It is the interface between an application and an in-process or out-of-process XR runtime that may handle frame composition, peripheral management, and more.

Specification and additional resources at [khronos.org/openxr](http://khronos.org/openxr)

### OpenXR API Overview

A high level overview of a typical OpenXR application including the order of function calls, creation of objects, session state changes, and the rendering loop.

---

#### OpenXR Action System Concepts

**Create action and action spaces**

- `xrCreateActionSet`
  
  `name = "gameplay"`

- `xrCreateAction`
  
  `actionSet="gameplay"`

  - `type = XR_INPUT_ACTION_TYPE_BOOLEAN`
  
  - `name = "teleport"`

- `xrCreateActionSpace`
  
  `action = "teleport_ray"`

**Set up interaction profile bindings**

- `xrSetInteractionProfileSuggestedBindings`

  ```
  /interaction_profiles/oculus/touch_controller
  "teleport": /user/hand/right/input/a/click
  "teleport_ray": /user/hand/right/input/trackpad/click
  /interaction_profiles/htc/vive_controller
  "teleport": /user/hand/right/input/trackpad/click
  "teleport_ray": /user/hand/right/input/trackpad/click
  ```

- `xrAttachSessionActionSets`

  ```
  session
  actionSets = { "gameplay", ... }
  ```

**Sync and get action states**

- `xrSyncActions`

  ```
  session
  activeActionSets = { "gameplay", ... }
  ```

- `xrGetActionStateBoolean("teleport_ray")`

  ```
  if (state.currentState) // button is pressed
  {
    xrLocateSpace (teleport_ray_space,
    stage_reference_space);
  }
  ```

OpenXR separates application actions such as Move, Jump, and Teleport from the input Trigger, Thumbstick, Button, etc. Actions are grouped into application-defined action sets that correspond to usage context (menu, gameplay, etc.). This simplifies support for different or future input devices and maximizes user accessibility.

Interaction profiles identify a collection of buttons and other input sources in a physical arrangement to allow applications and runtimes to coordinate action-to-input mapping. Runtimes bind actions to input devices based on application-supplied suggested bindings and other runtime-specific sources. This permits developers to customize to hardware they have tested, while making it possible to run on other hardware as supported by runtimes.

Syncing actions selects the active action set(s) to receive input, and updates the action states. Most input data is accessible with `xrGetActionState*` functions. Pose actions for tracked objects use "action spaces" and `xrLocateSpace` instead, for use like reference spaces.

---
OpenXR Fundamentals

### Traversing pointer chains [2.7.7]

typedef struct XrBaseInStructure
{
    XrStructureType type;
    const struct XrBaseInStructure* next;
} XrBaseInStructure;

typedef struct XrBaseOutStructure
{
    XrStructureType type;
    struct XrBaseOutStructure* next;
} XrBaseOutStructure;

---

Macros for version and header control

#### Version numbers [2.1, Appendix]

typedef uint64_t XR_VERSION;

Version numbers are encoded in 64 bits as follows:

- bits 63-48: Major version
- bits 47-32: Minor version
- bits 31-0: Patch version

#### Version macros

- #define XR_CURRENT_API_VERSION
- #define XR_MAKE_VERSION
- #define XR_VERSION_MAJOR
- #define XR_VERSION_MINOR
- #define XR_VERSION_PATCH
- #define XR_VERSION_X
- #define XR_VERSION Tháng
- #define XR_VERSION_X
- #define XR_VERSION_PATCH

---

#### Graphics API header control [Appendix]

<table>
<thead>
<tr>
<th>Compile Time Symbol</th>
<th>Graphics API</th>
</tr>
</thead>
<tbody>
<tr>
<td>XR_USE_GRAPHICS_API_OPENGL</td>
<td>OpenGL</td>
</tr>
<tr>
<td>XR_USE_GRAPHICS_API_OPENGL_ES</td>
<td>OpenGL ES</td>
</tr>
<tr>
<td>XR_USE_GRAPHICS_API_VULKAN</td>
<td>Vulkan</td>
</tr>
<tr>
<td>XR_USE_GRAPHICS_API_D3D11</td>
<td>Direct3D 11</td>
</tr>
<tr>
<td>XR_USE_GRAPHICS_API_D3D12</td>
<td>Direct3D 12</td>
</tr>
</tbody>
</table>

---

#### Window system header control [Appendix]

<table>
<thead>
<tr>
<th>Compile Time Symbol</th>
<th>Window System</th>
</tr>
</thead>
<tbody>
<tr>
<td>XR_USE_PLATFORM_WIN32</td>
<td>Microsoft Windows</td>
</tr>
<tr>
<td>XR_USE_PLATFORM_XLIB</td>
<td>X Window System Xlib</td>
</tr>
<tr>
<td>XR_USE_PLATFORM_XCB</td>
<td>X Window System Xcb</td>
</tr>
<tr>
<td>XR_USE_PLATFORM_WAYLAND</td>
<td>Wayland</td>
</tr>
<tr>
<td>XR_USE_PLATFORM_ANDROID</td>
<td>Android Native</td>
</tr>
</tbody>
</table>

---

Buffer size parameters [2.11]

Some functions refer to input/output buffers of parameters of the following form:

XrResult xrfunction(uint32_t* elementCapacityInput, uint32_t* elementCountOutput, float* elements);

Two-call idiom for buffer size parameters

First call xrfunction() with a valid elementCountOutput pointer (always required), elements = NULL, and elementCapacityInput = 0 to get the number of elements in the buffer; allocate sufficient space, then call xrfunction() again with the allocated buffer’s parameters.

---

### Macros for version and header control

#### Version numbers [2.1, Appendix]

typedef uint64_t XR_VERSION;

Version numbers are encoded in 64 bits as follows:

- bits 63-48: Major version
- bits 47-32: Minor version
- bits 31-0: Patch version

#### Version macros

- #define XR_CURRENT_API_VERSION
- #define XR_MAKE_VERSION(major, minor, patch) (((major) & 0xFF) << 24) | ((minor) & 0xFF) << 16) | (patch & 0xFF)
- #define XR_VERSION_MAJOR(version) ((version) & 0xFFFF)
- #define XR_VERSION_MINOR(version) ((version) >> 16) & 0xFFFF
- #define XR_VERSION_PATCH(version) ((version) >> 48) & 0xFFFF
- #define XR_VERSION_STR(version) [[2.7.7]]

---

#### Data types

**Color** [2.14]

Color components are linear (e.g., not sRGB), not alpha-premultiplied, in the range 0.0..1.0.

typedef struct XrColor4f {
    float r; float g; float b; float a;
} XrColor4f;

#### Coordinate system [2.15]

OpenXR uses a Cartesian right-handed coordinate system with an x, y, and z axis.

Points and directions can be represented using the following structure types with the following members:

- XrVector2f
  - Members x, y for distance in meters or 2D direction
- XrVector3f
  - Members x, y, z for distance in meters, or velocity or angular velocity
- XrVector4f
  - Members x, y, z, w for a 4D vector construct
- XrQuaternionf
  - Members x, y, z, w representing 3D orientation as a unit quaternion
  - Members orientation as a unit quaternion and position in meters

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

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

typedef struct XrVector4f {
    float x;
    float y;
    float z;
    float w;
} XrVector4f;

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

---

**Time**

#### XR_TIME[2.12.1]

A 64-bit integer representing a time relative to a runtime-dependent epoch. All simultaneous applications use the same epoch.

#### XR_DURATION [2.13]

A 64-bit signed integer representing a duration; the difference between two XRTime values.

**Special values:**

- #define XR_NO_DURATION 0
- #define XR_INFINITY_DURATION 0x7FFFFFFFFFFFFL

---

#### Convenience macros [2.8.3]

- #define XR_SUCCESS
- #define XR_FAILED
- #define XR_TIMEOUT
- #define XR_SESSION_LOSS
- #define XR_APPLICATION_LOSS
- #define XR_SWAPCHAIN_LOSS
- #define XR_INSTANCE_LOSS
- #define XR_SESSION_STOP

---

©2019 Khronos Group - Rev. 0719

www.khronos.org/openxr
OpenXR 1.0 Reference Guide

Instance lifecycle

API layers and extensions [2.7, 4.1]

API layers are inserted between the application and the runtime to hook API calls for logging, debugging, validation, etc. Extensions can expose new features or modify the behavior of existing functions. Both extensions and API layers are selected at XrInstance creation. To enable a layer, add its name to the enabledApiLayerNames member of XrInstanceCreateInfo. To enable an extension, add its name to the enabledExtensions member of XrInstanceCreateInfo.

typedef struct
XrExtent2Di {
    int32_t x;
    int32_t y;
} XrExtent2Di;

Members specify a rectangular area in meters if physical.

typedef struct
XrExtent2Df {
    float x;
    float y;
} XrExtent2Df;

Members specify a rectangular area in meters if physical.

typedef struct
XrRect2Di {
    int32_t t;
    int32_t w;
} XrRect2Di;

Members specify a rectangular area in meters if physical.

typedef struct
XrRect2Df {
    float t;
    float w;
} XrRect2Df;

Command function pointers [3.2]

XrResult xrGetInstanceProcAddr(XrInstance instance, const char* name, PFN_xrVoidFunction* function);

Instance lifecycle [4.2]

Call xrCreateInstance() to get an XrInstance handle. The instance manages the interface between the application and the OpenXR runtime.

XrResult xrCreateInstance(XrInstanceCreateInfo* createInfo, XrInstance* instance);

typedef struct
XrInstanceCreateInfo {
    XrStructureType type;
    uint32_t enabledApiLayerNames;
    char charExtensionNames[256];
} XrInstanceCreateInfo;

typedef struct
XrApiLayerProperties {
    XrStructureType type;
    void* next;
    char charName[448];
    XR_MAX_API_LAYER_DESCRIPTION_SIZE;
} XrApiLayerProperties;

typedef struct
XrExtensionProperties {
    XrStructureType type;
    void* next;
    char charName[448];
    XR_MAX_EXTENSION_NAME_SIZE;
} XrExtensionProperties;

XrResult xrEnumerateInstanceExtensionProperties(XrInstance instance, const char* name, XrExtensionProperties** properties);

typedef struct
XrExtensionProperties {
    XrStructureType type;
    void* next;
    char charName[448];
    XR_MAX_EXTENSION_NAME_SIZE;
} XrExtensionProperties;

XrResult xrEnumerateApiLayerProperties(XrInstance instance, XrApiLayerProperties** properties);

XrResult xrEnumerateInstanceExtensionProperties(XrInstance instance, const char* name, XrExtensionProperties** properties);

typedef struct
XrExtensionProperties {
    XrStructureType type;
    void* next;
    char charName[448];
    XR_MAX_EXTENSION_NAME_SIZE;
} XrExtensionProperties;

XrResult xrEnumerateApiLayerProperties(XrInstance instance, XrApiLayerProperties** properties);

Common types

Offsets, extents, and areas [2.16]

Members indicate offset in meters if physical.

typedef struct
XrOffset2Di {
    int32_t t;
    int32_t w;
} XrOffset2Di;

Members specify a rectangular area in meters if physical.

typedef struct
XrOffset2Df {
    int32_t t;
    int32_t w;
} XrOffset2Df;

Members specify a rectangular area in meters if physical.

typedef struct
XrExtent2Di {
    float width;
    float height;
} XrExtent2Di;

Members specify a rectangular area in meters if physical.

typedef struct
XrExtent2Df {
    float width;
    float height;
} XrExtent2Df;

FOV angles [2.17]

Angles are in radians from -π/2 to π/2.

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

XrResult xrCreateInstance(XrInstanceCreateInfo* createInfo, XrInstance* instance);

XrResult xrDestroyInstance(XrInstance instance);

System

Getting the XrSystemID [5.1-2]

XrResult xrGetSystemId(XrInstance instance, const char* systemId);

A return of XR_ERROR_FORM_FACTOR_UNAVAILABLE indicates the form factor is supported but temporarily unavailable; the application may retry xrGetSystemId.

typedef struct
XrSystemInfo {
    XrStructureType type;
    const void* next;
    XrFormFactor formFactor;
    XrStructureType formSystemId;
} XrSystemInfo;

typedef struct
XrSystemProperties {
    XrStructureType type;
    void* next;
    XrSystemId systemId;
    XrSystemId systemIdMD;
    XrSystemId systemIdS;
} XrSystemProperties;

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

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

System properties [5.3]

XrResult xrGetSystemProperties(XrInstance instance, XrSystemProperties** properties);

Getting system properties [5.3]

XrResult xrGetSystemProperties(XrInstance instance, XrSystemId systemId, XrSystemProperties** properties);

Semantic Paths and Path Tree

Path names and XrPath [6.1, 6.2]

Path name strings must contain only lower case a-z, digits 0-9, hyphen, underscore, period, or forward slash.

The XrPath is an atom that connects an application with a single path, within the context of a single instance. An XrPath is only shorthand for a well-formed path string, they have no explicit life cycle.

Path to string conversion [6.2.1]

XrResult xrStringToPath(XrInstance instance, XrResult value, char* buffer[XR_MAX_RESULT_STRING_SIZE]);

XrResult xrStructureTypeToString(XrInstance instance, XrStructureType type, char* buffer[XR_MAX_STRUCTURE_NAME_SIZE]);

Reserved paths [6.3.1]

"/user/"/left"/head"/right"/head"/gamepad"/treadmill"

Input/output subpaths [6.2.3-2]

Input source paths are of the form:

/identifier1/identifier2/identifier3

For extensions, the form is:

/extension/newidentifier_ext/newcomponent_ext

The path names for devices such as haptics follow this form:

/identifier1/identifier2/identifier3

Continued on next page >
Spaces

Working with spaces [7.3]

```
XrResult xrDestroySpace(XrSpace space);
XrResult xrLocateSpace(XrSpace space, XrSpace baseSpace, XrTime time, XrSpaceLocation* location);
```

typedef struct XrSpaceLocation {
    XrSpaceType type;
    void* next;
    XrSpaceLocationFlags locationFlags;
    XrPosef pose;
} XrSpaceLocation;

```
XrSpaceLocation:
    locationFlags: A bitwise OR of zero or more of:
        XrSPACE_LOCATION_ORIENTATION_VALID_BIT,
        XrSPACE_LOCATION_POSITION_VALID_BIT,
        XrSPACE_LOCATION_ORIENTATION_TRACKED_BIT,
        XrSPACE_LOCATION_POSITION_TRACKED_BIT
XrSpaceVelocity may be passed in using the next chain of
XrSpaceLocation to determine the velocity.
```

typedef struct XrSpaceVelocity {
    XrSpaceType type;
    void* next;
    XrSpaceVelocityFlags velocityFlags;
    XrVector3f linearVelocity;
    XrVector3f angularVelocity;
} XrSpaceVelocity;

Reference spaces [7.1]

```
XrResult xrEnumerateReferenceSpaces(XrSession session, uint32_t spaceCapacityInput, uint32_t* spaceCountOutput, XrReferenceSpaceType* spaces);
XrResult xrCreateReferenceSpace(XrSession session, const XrReferenceSpaceCreateInfo* createInfo, XrSpace* space);
```

typedef struct XrReferenceSpaceCreateInfo {
    XrSpaceType type;
    const void* next;
    XrReferenceSpaceType referenceSpaceType;
    XrPosef poseInReferenceSpace;
} XrReferenceSpaceCreateInfo;

Rendering [10]

Swatchpains [10.1]

```
XrResult xrEnumerateSwapchainFormats(XrSession session, uint32_t formatCapacityInput, uint32_t* formatCountOutput, XrSwapchainFormat* formats);
XrResult xrCreateSwapchain(XrSession session, const void* createInfo, XrSwapchain* swapchain);
```

typedef struct XrSwapchainCreateInfo {
    XrSpaceType type;
    const void* next;
    XrSwapchainUsageFlags usageFlags;
    int64_t format;
    int64_t sampleCount, width, height, mipCount;
    uint32_t arraySize;
    uint32_t mipCount;
    void* next;
} XrSwapchainCreateInfo;

```
XrResult xrGetReferenceSpaceBoundsRect(XrSession session, XrReferenceSpaceType referenceSpaceType, XrExtent2Df* bounds);
```

An XrEventDataReferenceSpaceChangePending event is sent
to the application when the origin (and possibly bounds) of a
reference space is changing:

```
XrResult xrCreateActionSpace(XrSession session, const XrActionSpaceCreateInfo* createInfo, XrSpace* space);
```

typedef struct XrActionSpaceCreateInfo {
    XrSpaceType type;
    const void* next;
    XrAction action;
    XrPath subactionPath;
    XrPosef poseInActionSpace;
} XrActionSpaceCreateInfo;

```
XrResult xrEnumerateSwapchainImages(XrSwapchain swapchain, uint32_t imageCapacityInput, uint32_t* imageCountOutput, XrSwapchainImageBaseHeader* images);
```

typedef struct XrSwapchainImageBaseHeader {
    XrSpaceType type;
    const void* next;
    XrSwapchainImageBaseHeader type;
    uint32_t imageIndex;
} XrSwapchainImageBaseHeader;

```
XrResult xrEnumerateSwapchainImages(XrSwapchain swapchain, XrSwapchainImageBaseHeader* images, uint32_t* index);
```

```
XrResult xrGetSwapchainImageInfo(XrSwapchain swapchain, uint32_t imageIndex, XrSwapchainImageInfo* info);
XrResult xrGetSwapchainImageInfo(XrSwapchain swapchain, uint32_t imageIndex, uint32_t* index);
```

```
XrResult xrAcquireSwapchainImage(XrSwapchain swapchain, const XrSwapchainImageAcquireInfo* acquireInfo, uint32_t* index);
```

```
XrResult xrWaitSwapchainImage(XrSwapchain swapchain, XrSwapchainImageWaitInfo* waitInfo);
```

```
XrResult xrReleaseSwapchainImage(XrSwapchain swapchain, const XrSwapchainImageReleaseInfo* releaseInfo);
```

Continued on next page >
OpenXR session life cycle [9.3]

An XrSession proceeds through a number of states based on application requests, runtime operations, and user actions. The following diagram shows the session state machine. The state boxes are labeled with a name that is associated with an XrSessionState value.

Environment Blend Mode [10.5.7]

XrResult xRunEnumerateEnvironmentBlendModes(XrInstance instance, XrSystemId systemId, XrCompositionLayerBaseHeader* layers);

Environment Blend Mode values: XR_ENVIRONMENT_BLEND_MODE_OPAQUE, XR_ENVIRONMENT_BLEND_MODE_ADDITIVE, XR_ENVIRONMENT_BLEND_MODE_ALPHA_BLEND

Notes

---

©2019 Khronos Group - Rev. 0719
Compositing

Compositing [10.5]

Composition layers are submitted by the application via the xrenFrame call. All composition layers to be drawn must be submitted with every xrenFrame call. Composition layers are drawn in the same order as they are specified in via XrFrameEndInfo, with the 0th layer drawn first.

typedef struct XrCompositionLayerBaseHeader {
  XrStructureType type;
  const void* next;
  XrCompositionLayerFlags flagsFlags;
  XrSpace space;
} XrCompositionLayerBaseHeader;

layerFlags: A bitwise OR of XR_COMPOSITION_LAYER_Y_BIT where Y may be: CORRECT, CHROMATIC_ABERRATION, BLEND_TEXTURE_SOURCE_ALPHA

type: XR_TYPE_COMPOSITION_LAYER_X where X may be: PROJECTION, QUAD, CUBE_KHR, CYLINDER_KHR, EQUIRECT_KHR

next: NULL or a pointer to an extension-specific structure:
  XrCompositionLayerColorModulationInfoKHR if the Xr_KHR_composition_layer_color_modulation_extension is enabled, or
  XrCompositionLayerDepthInfoKHR if Xr_KHR_composition_layer_depth is enabled

typedef struct XrSwapchainSubImage {
  XrSwapchain swapchain;
  XrRect2Di imageRect;
  uint32_t imageArrayIndex;
} XrSwapchainSubImage;

typedef struct XrCompositionLayerProjection {
  XrSpace space;
  const void* next;
  XrPosef pose;
  char* path;
} XrCompositionLayerProjection;

typedef struct XrCompositionLayerProjectionView {
  XrSpace space;
  const void* next;
  XrPosef pose;
} XrCompositionLayerProjectionView;

XR_KHR_composition_layer_cube [12.25]

This extension adds an additional layer type that enables direct sampling from cubemaps.

typedef struct XrCompositionLayerCubeKHR {
  XrSpace space;
  float farZ;
  float nearZ;
  float minDepth;
  float maxDepth;
} XrCompositionLayerCubeKHR;

Input and Haptics: Actions

Actions are created at initialization time and later used to request input device state, create action spaces, or control haptic events.

Action sets [11.2]

XrResult xrCreateActionSet(XrInstance instance, const XrActionSetCreateInfo* createInfo, XrActionSet* actionSet);

typedef struct XrActionSetCreateInfo {
  XrStructureType type;
  const void* next;
  XrActionSetName* actionName;
  const uint32_t actionType;
  const uint32_t countSubactionPaths;
  const uint32_t queueId;
  const XrPath* path;
} XrActionSetCreateInfo;

XrResult xrDestroyActionSet(XrActionSet actionSet);

Actions [11.3]

XrResult xrCreateAction(XrActionSet actionSet, const XrActionCreateInfo* createInfo, XrAction* action);

typedef struct XrActionCreateInfo {
  XrStructureType type;
  const void* next;
  const char* actionName;
  const uint32_t countSubactionPaths;
  const uint32_t queueId;
  const XrPath* path;
} XrActionCreateInfo;

Suggested Bindings [11.4]

Applications need to provide default bindings for their actions to runtimes so that input data can be mapped appropriately to the application’s actions. The bindings suggested by this system are only a hint to the runtime.

XrResult xrSuggestInterationProfileSuggestedBinding(XrInstance instance, const XrInteractionProfileSuggestedBinding* suggestedBindings);

typedef struct XrInteractionProfileSuggestedBinding {
  XrStructureType type;
  const void* next;
  const XrPath* path;
} XrInteractionProfileSuggestedBinding;

XrResult xrCreateSuggestedBinding(XrAction action, const XrPath* path);

typedef struct XrActionSuggestedBinding {
  XrStructureType type;
  const void* next;
} XrActionSuggestedBinding;
Extensions [12]

Extension naming convention [2.6]

XR_KHR_* - Kronos-created extensions supported by multiple vendors
XR_EXT_* - extensions supported by multiple vendors, possibly API restricted

XR_KHR_convert timespec time [12.9]
Enabling this extension makes the following available.

XrResult xrConvertTimeSpecToTimeKHR(XrInstance instance, const timespec* timespec, uint64_t* time);

XR_KHR_D3D11_enable [12.11]
Support the D3D 11 graphics API in an OpenXR runtime.

XrResult xrGetD3D11GraphicsRequirementsKHR(XrInstance instance, VkSurfaceFormatKHR format);

typedef struct {XrGraphicsBindingD3D11KHR * binding; VkSurfaceFormatKHR format; XrResult result;} D3D11GraphicsRequirementsKHR;

typedef struct XRGraphicsRequirementsD3D11KHR {
    XrStructureType type;
    void* next;
    D3D11Device* device;
} XRGraphicsRequirementsD3D11KHR;

Input Action State Synchronization [11.7]
XrResult xrSyncActions(XrSession session, const XrActionsSyncInfo* info);

typedef struct {XrStructureType type; void* next; XrAction action; } XRActionSetVector;

Output Actions and Haptics [11.6]
XrResult xrPlayHapticFeedback(XrActionsSyncInfo* info, const XrHapticFeedbackInfo* info);

typedef struct {XrStructureType type; float* frequency; float* duration; } XRInputSourceLocalizedNameGetInfo;

Action Sources [11.8]
XrResult xrEnumerateBoundSourcesForAction(XrSession session, const XrBoundSourceInfoKHR* info, uint32_t* count);

typedef struct {XrStructureType type; void* next; } XRInputSource;

Action Sources [11.8]
XrResult xrEnumerateBoundSourcesForAction(XrSession session, const XrBoundSourceInfoKHR* info, uint32_t* count);

typedef struct {XrStructureType type; void* next; } XRInputSource;

Action Sources [11.8]
XrResult xrEnumerateBoundSourcesForAction(XrSession session, const XrBoundSourceInfoKHR* info, uint32_t* count);

typedef struct {XrStructureType type; void* next; } XRInputSource;

Action Sources [11.8]
XrResult xrEnumerateBoundSourcesForAction(XrSession session, const XrBoundSourceInfoKHR* info, uint32_t* count);

typedef struct {XrStructureType type; void* next; } XRInputSource;

Action Sources [11.8]
XrResult xrEnumerateBoundSourcesForAction(XrSession session, const XrBoundSourceInfoKHR* info, uint32_t* count);

typedef struct {XrStructureType type; void* next; } XRInputSource;
typedef struct XrSwapchainImageOpenGLESKHR {
  XrStructureType type;
  void* next;
  uint32_t image;
} XrSwapchainImageOpenGLESKHR;

XrResult xrGetOpenGLGraphicsRequirementsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  XrGraphicsRequirementsOpenGLESKHR* graphicsRequirements);

typedef struct XrSwapchainImageOpenGLESKHR {
  XrStructureType type;
  void* next;
  uint32_t image;
} XrSwapchainImageOpenGLESKHR;

XrSwapchainImageOpenGLESKHR* xrGetOpenGLGraphicsRequirementsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

XrKHR_vulkan_swapchain_format_list [12.18]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* image;
  uint32_t* bufferCapacityInput;
  uint32_t* bufferCountOutput, char* buffer;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

Xr_KHR_visiblity_mask [12.16]
This extension enables the following:
XrResult xrGetVisibilityMaskKHR( 
  XrSession session, 
  XrViewConfigurationType viewConfigurationType, 
  uint32_t viewIndex, 
  XrVisibilityMaskTypeKHR visibilityMaskType, 
  XrVisibilityMaskKHR* visibilityMask);

visibilityMask::
  XR_VISIBILITY_MASK_TYPE_X_KHR where X may be: 
  HIDDEN_TRIANGLE_MESH, VISIBLE_TRIANGLE_MESH, 
  LINE_LOOP

XrVulkanSwapchainFormatListCreateInfoKHR* xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* imageCountOutput, char* imageCountOutput);

Xr_EVT_visibility_mask_typeKHR [12.17]
Support the Vulkan graphics API in an OpenXR runtime in 
addition to those functions and structs shown under Sessions on page 6 of this reference guide.
XrResult xrGetVulkanGraphicsDeviceKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  VkInstance vkInstance, 
  VkPhysicalDevice* vkPhysicalDevice);

XrResult xrGetVulkanInstanceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

Xr_KHR_win32_convert_performance_counter_time [12.19]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* imageCountOutput, char* imageCountOutput;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrConvertWin32PerformanceCounterToTimeKHR( 
  XrInstance instance, 
  const LARGE_INTEGER* performanceCounter, 
  const VkFormat* format);

XrResult xrConvertTimeToWin32PerformanceCounterKHR( 
  XrInstance instance, 
  const VkFormat* format, 
  const LARGE_INTEGER* time);

Xr_VisibilityMaskKHR* xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* imageCountOutput, char* imageCountOutput);

Xr_KHR_vulkan_swapchain_format_list [12.18]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* image;
  uint32_t* bufferCapacityInput;
  uint32_t* bufferCountOutput, char* buffer;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

Xr_KHR_vulkan_swapchain_format_list [12.18]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* image;
  uint32_t* bufferCapacityInput;
  uint32_t* bufferCountOutput, char* buffer;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

Xr_KHR_vulkan_swapchain_format_list [12.18]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* image;
  uint32_t* bufferCapacityInput;
  uint32_t* bufferCountOutput, char* buffer;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

Xr_KHR_vulkan_swapchain_format_list [12.18]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* image;
  uint32_t* bufferCapacityInput;
  uint32_t* bufferCountOutput, char* buffer;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);

Xr_KHR_vulkan_swapchain_format_list [12.18]
This extension enables the following:
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
  XrStructureType type;
  const void* next;
  uint32_t* image;
  uint32_t* bufferCapacityInput;
  uint32_t* bufferCountOutput, char* buffer;
} XrVulkanSwapchainFormatListCreateInfoKHR;

XrResult xrGetVulkanDeviceExtensionsKHR( 
  XrInstance instance, 
  XrSystemId systemId, 
  uint32_t* bufferCountOutput, char* buffer);