OpenXR 1.0 Reference Guide

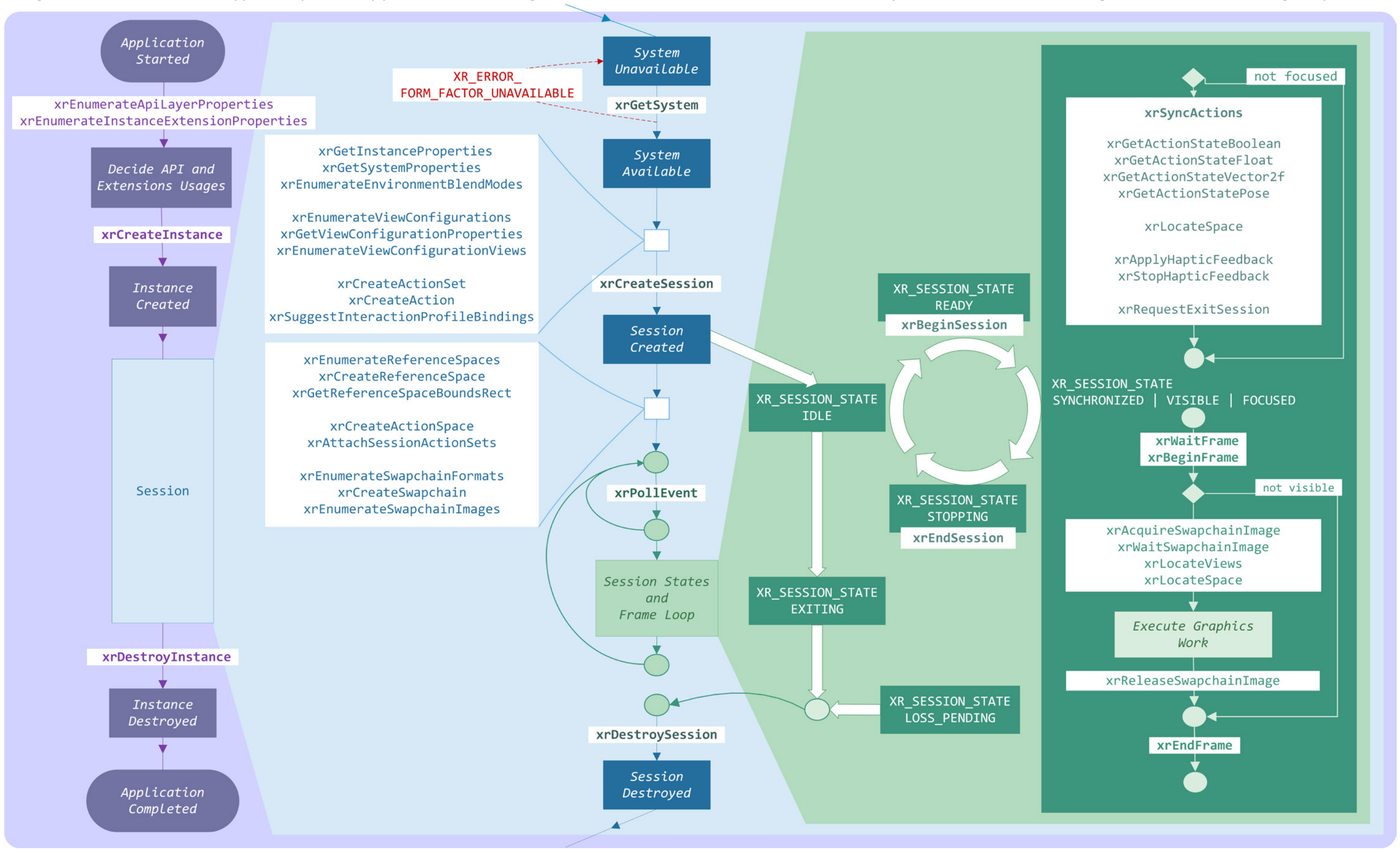
OpenXR™ is a cross-platform API that enables a continuum of real-and-virtual combined environments generated by computers through human-machine interaction and is inclusive of the technologies associated with virtual reality, augmented reality, and mixed reality. It is the interface between an application 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**

Color-coded names as follows: **function names** and **structure names**.
[n.n.n] Indicates sections and text in the OpenXR 1.0 specification.
● Indicates content that pertains to an extension.

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 [11.1]

Create action and action spaces

xrCreateActionSet
name = "gameplay"

xrCreateAction
actionSet="gameplay"
name = "teleport"
type = XR_INPUT_ACTION_TYPE_BOOLEAN
actionSet="gameplay"
name = "teleport_ray"
type = XR_INPUT_ACTION_TYPE_POSE

xrCreateActionSpace
action = "teleport_ray"

Set up interaction profile bindings

xrSuggestInteractionProfileBindings
/interaction_profiles/oculus/touch_controller
"teleport": /user/hand/right/input/a/click
"teleport_ray": /user/hand/right/input/aim/pose

/interaction_profiles/htc/vive_controller
"teleport": /user/hand/right/input/trackpad/click
"teleport_ray": /user/hand/right/input/aim/pose

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;
```

Buffer size parameters [2.11]

Some functions refer to input/output buffers with 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 **XrVersion**;

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
    XR_MAKE_VERSION(1, 0, 0)

#define XR_MAKE_VERSION(major, minor, patch)
    ((((major) & 0xffffULL) << 48) |
    (((minor) & 0xffffULL) << 32) | ((patch) & 0xffffffffULL))

#define XR_VERSION_MAJOR(version)
    (uint16_t) (((uint64_t)(version) >> 48) & 0xffffULL)

#define XR_VERSION_MINOR(version)
    (uint16_t) (((uint64_t)(version) >> 32) & 0xffffULL)

#define XR_VERSION_PATCH(version)
    (uint32_t) ((uint64_t)(version)  & 0xffffffffULL)
```

Graphics API header control [Appendix]

Compile Time Symbol	API
XR_USE_GRAPHICS_API_OPENGL	OpenGL
XR_USE_GRAPHICS_API_OPENGL_ES	OpenGL ES
XR_USE_GRAPHICS_API_VULKAN	Vulkan
XR_USE_GRAPHICS_API_D3D11	Direct3D 11
XR_USE_GRAPHICS_API_D3D12	Direct3D 12

Window system header control [Appendix]

Compile Time Symbol	Window System
XR_USE_PLATFORM_WIN32	Microsoft Windows
XR_USE_PLATFORM_XLIB	X Window System Xlib
XR_USE_PLATFORM_XCB	X Window System Xcb
XR_USE_PLATFORM_WAYLAND	Wayland
XR_USE_PLATFORM_ANDROID	Android Native

Threading behavior [2.3]

OpenXR functions generally support being called from multiple threads with a few exceptions:

- The *handle* parameter and any child handles that will be destroyed by a destroy function must be externally synchronized.
  - 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**
- Calls to **xrWaitFrame** for a given XrSession must be externally synchronized.

XR_KHR_android_thread_settings [12.3]

○ If enabled, this extension allows the application to specify the Android thread type.

```
XrResult xrSetAndroidApplicationThreadKHR(
    XrSession session,
    XrAndroidThreadTypeKHR threadType,
    uint32_t threadId);
```

threadType: XR_ANDROID_THREAD_TYPE_*X*_KHR
where *X* may be:
APPLICATION_MAIN, APPLICATION_WORKER,
RENDERER_MAIN, RENDERER_WORKER

Time

XrTime [2.12.1]

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

XrDuration [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_INFINITE_DURATION 0x7fffffffffffffffLL
```

Data types

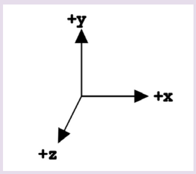
Color [2.14]

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

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

XrResult return codes [2.8]

API commands return values of type XrResult. Negative values are error codes, while non-negative (≥0) are success codes.

Success codes

XR_SUCCESS	XR_TIMEOUT_EXPIRED
XR_SESSION_LOSS_PENDING	XR_EVENT_UNAVAILABLE
XR_SESSION_NOT_FOCUSED	XR_FRAME_DISCARDED
XR_SPACE_BOUNDS_UNAVAILABLE	

Error codes

XR_ERROR_*X* where *X* may be:

ACTION_TYPE_MISMATCH
ACTIONSET_NOT_ATTACHED
ACTIONSETS_ALREADY_ATTACHED
ANDROID_THREAD_SETTINGS_FAILURE_KHR
ANDROID_THREAD_SETTINGS_ID_INVALID_KHR
API_LAYER_NOT_PRESENT
API_VERSION_UNSUPPORTED
CALL_ORDER_INVALID
ENVIRONMENT_BLEND_MODE_UNSUPPORTED
EXTENSION_NOT_PRESENT
FEATURE_UNSUPPORTED
FILE_ACCESS_ERROR
FILE_CONTENTS_INVALID
FORM_FACTOR_UNAVAILABLE
FORM_FACTOR_UNSUPPORTED
FUNCTION_UNSUPPORTED
GRAPHICS_DEVICE_INVALID
HANDLE_INVALID
INDEX_OUT_OF_RANGE
INITIALIZATION_FAILED
INSTANCE_LOST
LAYER_INVALID
LAYER_LIMIT_EXCEEDED
LIMIT_REACHED
LOCALIZED_NAME_DUPLICATED
LOCALIZED_NAME_INVALID
NAME_DUPLICATED
NAME_INVALID
OUT_OF_MEMORY
PATH_COUNT_EXCEEDED
PATH_FORMAT_INVALID
PATH_INVALID
PATH_UNSUPPORTED
POSE_INVALID
REFERENCE_SPACE_UNSUPPORTED
RUNTIME_FAILURE
SESSION_LOST
SESSION_NOT_READY
SESSION_NOT_RUNNING
SESSION_NOT_STOPPING
SESSION_RUNNING
SIZE_INSUFFICIENT
SWAPCHAIN_FORMAT_UNSUPPORTED
SWAPCHAIN_RECT_INVALID
SYSTEM_INVALID
TIME_INVALID
VALIDATION_FAILURE
VIEW_CONFIGURATION_TYPE_UNSUPPORTED

○ **XR_KHR_android_thread_settings**

This extension enables the following additional error codes:

XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR
XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR

Convenience macros [2.8.3]

```
#define XR_SUCCEEDED(result) ((result) >= 0)
    XR_SUCCEEDED is true for non-negative codes.

#define XR_FAILED(result) ((result) < 0)
    XR_FAILED is true for negative codes.

#define XR_UNQUALIFIED_SUCCESS(result) ((result) == 0)
    XR_UNQUALIFIED_SUCCESS is true for 0 (XR_SUCCESS) only.
```

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.

```
XrResult xrEnumerateApiLayerProperties(
    uint32_t propertyCapacityInput,
    uint32_t* propertyCountOutput,
    XrApiLayerProperties* properties);

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;

XrResult xrEnumerateInstanceExtensionProperties(
    const char* layerName,
    uint32_t propertyCapacityInput,
    uint32_t* propertyCountOutput,
    XrExtensionProperties* properties);

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

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(
    const XrInstanceCreateInfo* createInfo,
    XrInstance* instance);

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;
```

createFlags must be 0

```
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;

XrResult xrDestroyInstance(XrInstance instance);
```

XR_KHR_android_create_instance [12.1]

This extension enables the following:

```
typedef struct XrInstanceCreateInfoAndroidKHR {
    XrStructureType type;
    const void* next;
    void* applicationVM;
    void* applicationActivity;
} XrInstanceCreateInfoAndroidKHR;
```

Instance information [4.3]

```
XrResult xrGetInstanceProperties(XrInstance instance,
    XrInstanceProperties* instanceProperties);

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

XrEventDataInstanceLossPending [4.4.2]

Receiving this structure predicts a session loss at *lossTime*. The application should call **xrDestroyInstance** and release instance resources.

```
typedef struct XrEventDataInstanceLossPending {
    XrStructureType type;
    const void* next;
    XrTime lossTime;
} XrEventDataInstanceLossPending;
```

Common types

Offsets, extents, and areas [2.16]

Members indicate offset in meters if physical.

```
typedef struct XrOffset2Df {
    float x;
    float y;
} XrOffset2Df;

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

Members specify a rectangular area in meters if physical.

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

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

Members specify a rectangular area in meters if physical.

```
typedef struct XrRect2Df {
    XrOffset2Df offset;
    XrExtent2Df extent;
} XrRect2Df;

typedef struct XrRect2Di {
    XrOffset2Di offset;
    XrExtent2Di extent;
} XrRect2Df;
```

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;
```

Boolean type [2.19]

The only valid values are XR_TRUE or XR_FALSE.

```
typedef uint32_t XrBool32;
```

Event polling [2.20.1]

The application is expected to allocate an event queue of type XrEventDataBuffer and periodically call **xrPollEvent**. If the event queue overflows, **xrPollEvent** will return the **XrEventDataEventsLost** event

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

XrResult xrPollEvent(XrInstance instance,
    XrEventDataBuffer*eventData);

typedef struct XrEventDataBaseHeader {
    XrStructureType type;
    const void* next;
} XrEventDataBaseHeader;

typedef struct XrEventDataEventsLost {
    XrStructureType type;
    const void* next;
    uint32_t lostEventCount;
} XrEventDataEventsLost;
```

Type to string conversions [4.5]

```
XrResult xrResultToString(XrInstance instance,
    XrResult value,
    char buffer[XR_MAX_RESULT_STRING_SIZE]);

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

System

Getting the XrSystemID [5.1-2]

```
XrResult xrGetSystem(XrInstance instance,
    const XrSystemGetInfo* getInfo, XrSystemId* systemId);
```

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

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

formfactor: XR_FORM_FACTOR_X where X may be:
HEAD_MOUNTED_DISPLAY, HANDHELD_DISPLAY

Getting system properties [5.3]

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

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

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

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

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. As 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,
    const char* pathString, XrPath* path);

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

Reserved paths [6.3.1]

/user/hand/left
/user/hand/right
/user/head
/user/gamepad
/user/treadmill

Input/output subpaths [6.3.2-3]

Input source paths are of the form:

.../input/<*identifier*>[_<*location*>][/<*component*>]

For extensions, the form is:

.../input/*newidentifier_ext*/*newcomponent_ext*

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

.../output/<*output_identifier*>[_<*location*>]

Continued on next page >

Semantic Paths / Path Tree (continued)

Standard values for identifier

trackpad thumbstick joystick
trigger pedal throttle
trackball thumbrest system
shoulder squeeze wheel

dpad_*X* where *X* may be: up, down, left, right

diamond_*X* where *X* may be: up, down, left, right

a, b, x, y, start, home, end, select

volume_up, volume_down, mute_mic, play_pause, menu

Standard pose identifiers

grip aim

Standard locations

left left_upper left_lower upper
right right_upper right_lower lower

Standard components

click force twist x, y
touch value pose

Standard output identifier

haptic

Interaction profile paths [6.4]

An interaction profile identifies a collection of buttons and other input sources, and is of the form:

/interaction_profiles/<*vendor_name*>/<*type_name*>

Paths supported in the core 1.0 release

/interaction_profiles/khr/simple_controller
/interaction_profiles/google/daydream_controller
/interaction_profiles/htc/vive_controller
/interaction_profiles/htc/vive_pro
/interaction_profiles/microsoft/motion_controller
/interaction_profiles/microsoft/xbox_controller
/interaction_profiles/oculus/go_controller
/interaction_profiles/oculus/touch_controller
/interaction_profiles/valve/index_controller

View configurations [8]

```
XrResult xrEnumerateViewConfigurations(
    XrInstance instance, XrSystemId systemId,
    uint32_t viewConfigurationTypeCapacityInput,
    uint32_t* viewConfigurationTypeCountOutput,
    XrViewConfigurationType* viewConfigurationTypes);
```

viewConfigurationTypes:
XR_VIEW_CONFIGURATION_TYPE_PRIMARY_MONO,
XR_VIEW_CONFIGURATION_TYPE_PRIMARY_STEREO

```
XrResult xrGetViewConfigurationProperties(
    XrInstance instance, XrSystemId systemId,
    XrViewConfigurationType viewConfigurationType,
    XrViewConfigurationProperties*
        configurationProperties);
```

```
typedef struct XrViewConfigurationProperties {
    XrStructureType type;
    void* next;
    XrViewConfigurationType viewConfigurationType;
    XrBool32 fovMutable;
} XrViewConfigurationProperties;
```

```
XrResult xrEnumerateViewConfigurationViews(
    XrInstance instance, XrSystemId systemId,
    XrViewConfigurationType viewConfigurationType,
    uint32_t viewCapacityInput, uint32_t*
    viewCountOutput, XrViewConfigurationView* views);
```

```
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;
```

Spaces

Working with spaces [7.3]

```
XrResult xrDestroySpace(XrSpace space);
```

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

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

locationFlags: A bitwise OR of zero or more of
XR_SPACE_LOCATION_ORIENTATION_VALID_BIT,
XR_SPACE_LOCATION_POSITION_VALID_BIT,
XR_SPACE_LOCATION_ORIENTATION_TRACKED_BIT,
XR_SPACE_LOCATION_POSITION_TRACKED_BIT

XrSpaceVelocity may be passed in using the next chain of XrSpaceLocation to determine the velocity.

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

velocityFlags: A bitwise OR of zero or more of
XR_SPACE_VELOCITY_LINEAR_VALID_BIT,
XR_SPACE_VELOCITY_ANGULAR_VALID_BIT

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 {
    XrStructureType type;
    const void* next;
    XrReferenceSpaceType referenceSpaceType;
    XrPosef poseInReferenceSpace;
} XrReferenceSpaceCreateInfo;
```

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

referenceSpaceType:
XR_REFERENCE_SPACE_TYPE_VIEW,
XR_REFERENCE_SPACE_TYPE_LOCAL,
XR_REFERENCE_SPACE_TYPE_STAGE

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

```
typedef
struct XrEventDataReferenceSpaceChangePending {
    XrStructureType type;
    const void* next;
    XrSession session;
    XrReferenceSpaceType referenceSpaceType;
    XrTime changeTime;
    XrBool32 poseValid;
    XrPosef poseInPreviousSpace;
} XrEventDataReferenceSpaceChangePending;
```

Action spaces [7.2]

An XrSpace handle for a pose action is created using **xrCreateActionSpace**, by specifying the chosen pose action and an optional transform from its natural origin. Examples of 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

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

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

Rendering [10]

Swapchains [10.1]

```
XrResult xrEnumerateSwapchainFormats(
    XrSession session, uint32_t formatCapacityInput,
    uint32_t* formatCountOutput, int64_t* formats);
```

Runtimes should support R8G8B8A8 and R8G8B8A8 sRGB formats. With OpenGL-based graphics APIs, the texture formats correspond to OpenGL internal formats. With Direct3D-based graphics APIs, **xrEnumerateSwapchainFormats** never returns typeless formats. Only concrete formats are returned or may be specified by applications for swapchain creation.

```
XrResult xrCreateSwapchain(XrSession session,
    const XrSwapchainCreateInfo* createInfo,
    XrSwapchain* swapchain);
```

```
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;
```

createFlags: A bitwise OR of zero or more of
XR_SWAPCHAIN_CREATE_PROTECTED_CONTENT_BIT,
XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT

usageFlags: A bitwise OR of zero or more of
XR_SWAPCHAIN_USAGE_*X*_BIT where *X* may be:
COLOR_ATTACHMENT,
DEPTH_STENCIL_ATTACHMENT, UNORDERED_ACCESS,
TRANSFER_SRC, TRANSFER_DST, SAMPLED,
MUTABLE_FORMAT

sampleCount, width, height, mipcount: Must not be 0

faceCount: 6 (for cubemaps) or 1

arraySize: Must not be 0: 1 is for a 2D image

```
XrResult xrDestroySwapchain(XrSwapchain swapchain);
```

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

```
typedef struct XrSwapchainImageBaseHeader {
    XrStructureType type;
    void* next;
} XrSwapchainImageBaseHeader;
```

type: XR_TYPE_SWAPCHAIN_IMAGE_*X*_KHR where *X* may be: OPENGL, OPENGL_ES, VULKAN, D3D11, D3D12

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

```
typedef struct XrSwapchainImageAcquireInfo {
    XrStructureType type;
    const void* next;
} XrSwapchainImageAcquireInfo;
```

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

```
typedef struct XrSwapchainImageWaitInfo {
    XrStructureType type;
    const void* next;
    XrDuration timeout;
} XrSwapchainImageWaitInfo;
```

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

```
typedef struct XrSwapchainImageReleaseInfo {
    XrStructureType type;
    const void* next;
} XrSwapchainImageReleaseInfo;
```

Continued on next page >

Rendering (continued)

[12.2] XR_KHR_android_surface_swapchain
This extension enables the Android swapchain function:

```
XrResult xrCreateSwapchainAndroidSurfaceKHR(
    XrSession session, const XrSwapchainCreateInfo* info,
    XrSwapchain* swapchain, jobject* surface);
```

[12.18] XR_KHR_vulkan_swapchain_format_list
Enables the Vulkan VK_KHR_image_format_list extension.

```
typedef struct XrVulkanSwapchainFormatListCreateInfoKHR {
    XrStructureType type;
    const void* next;
    uint32_t viewFormatCount;
    const VkFormat* viewFormats;
} XrVulkanSwapchainFormatListCreateInfoKHR;
```

View and Projection State [10.2]

```
XrResult xrLocateViews(XrSession session,
    const XrViewLocateInfo* viewLocateInfo,
    XrViewState* viewState, uint32_t viewCapacityInput,
    uint32_t* viewCountOutput, XrView* views);
```

```
typedef struct XrViewLocateInfo {
    XrStructureType type;
    const void* next;
    XrViewConfigurationType viewConfigurationType;
    XrTime displayTime;
    XrSpace space
} XrViewLocateInfo;
```

```
typedef struct XrView {
    XrStructureType type;
    void* next;
    XrPosef pose;
    XrFovf fov;
} XrView;
```

```
typedef struct XrViewState {
    XrStructureType type;
    void* next;
    XrViewStateFlags viewStateFlags;
} XrViewState;
```

viewStateFlags: A bitwise OR of zero or more of XR_VIEW_STATE_*X*_BIT where *X* may be: ORIENTATION_VALID, POSITION_VALID, ORIENTATION_TRACKED, POSITION_TRACKED

Frame Waiting [10.4]

```
XrResult xrWaitFrame(XrSession session,
    const XrFrameWaitInfo* frameWaitInfo,
    XrFrameState* frameState);
```

```
typedef struct XrFrameWaitInfo {
    XrStructureType type;
    const void* next;
} XrFrameWaitInfo;
```

```
typedef struct XrFrameState {
    XrStructureType type;
    void* next;
    XrTime predictedDisplayTime;
    XrDuration predictedDisplayPeriod;
    XrBool32 shouldRender;
} XrFrameState;
```

Frame Submission [10.5]

```
XrResult xrBeginFrame(XrSession session,
    const XrFrameBeginInfo* frameBeginInfo);
```

```
typedef struct XrFrameBeginInfo {
    XrStructureType type;
    const void* next;
} XrFrameBeginInfo;
```

```
XrResult xrEndFrame(XrSession session,
    const XrFrameEndInfo* frameEndInfo);
```

```
typedef struct XrFrameEndInfo {
    XrStructureType type;
    const void* next;
    XrTime displayTime;
    XrEnvironmentBlendMode environmentBlendMode;
    uint32_t layerCount;
    const XrCompositionLayerBaseHeader* const* layers;
} XrFrameEndInfo;
```

layers: A pointer to an array of Projection and/or Quad types, or optionally:

[12.5] If XR_KHR_composition_layer_cube is enabled, then struct XrCompositionLayerCubeKHR can be used.

[12.6] If XR_KHR_composition_layer_cylinder is enabled, then struct XrCompositionLayerCylinderKHR can be used.

[12.8] If XR_KHR_composition_layer_equirect is enabled, then struct XrCompositionLayerEquirectKHR can be used.

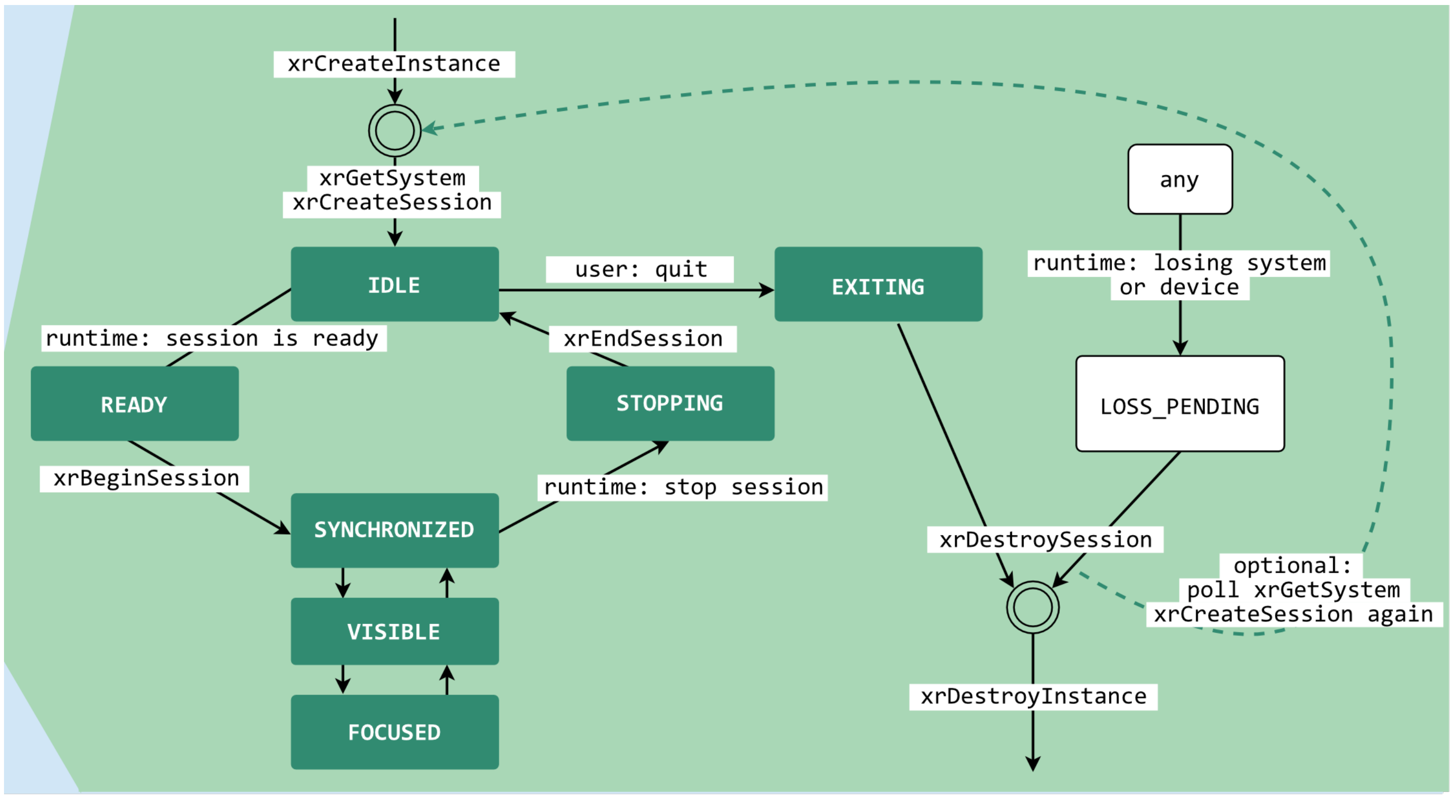
Environment Blend Mode [10.5.7]

```
XrResult xrEnumerateEnvironmentBlendModes(
    XrInstance instance, XrSystemId systemId,
    XrViewConfigurationType viewConfigurationType,
    uint32_t environmentBlendModeCapacityInput,
    uint32_t* environmentBlendModeCountOutput,
    XrEnvironmentBlendMode* environmentBlendModes);
```

Populates an array of *XrEnvironmentBlendMode* values: XR_ENVIRONMENT_BLEND_MODE_*X* where *X* may be: OPAQUE, ADDITIVE, ALPHA_BLEND

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.



Notes

Session [9]

Session lifecycle [9.1]

```
XrResult xrCreateSession(XrInstance instance,
    const XrSessionCreateInfo* createInfo,
    XrSession* session);

typedef struct XrSessionCreateInfo {
    XrStructureType type;
    const void* next;
    XrSessionCreateFlags createFlags;
    XrSystemId systemId;
} XrSessionCreateInfo;
```

createFlags must be 0

next: A pointer to an instance of XrGraphicsBinding*X* where *X* may be: D3D12KHR, D3D11KHR, OpenGLESAndroidKHR, OpenGLWaylandKHR, OpenGLXcbKHR, OpenGLXlibKHR, OpenGLWin32KHR, VulkanKHR

Using Graphics APIs in runtimes

Use extensions to enable access to OpenGL, OpenGL ES, Vulkan, and Direct3D 11 and 12 graphics APIs. The extended functions for using Vulkan are shown below. For others, see Extensions on page 7 of this reference guide.

[12.13] Enabled with XR_KHR_vulkan_enable

```
XrResult xrGetVulkanGraphicsRequirementsKHR(
    XrInstance instance, XrSystemId systemId,
    XrGraphicsRequirementsVulkanKHR*
        graphicsRequirements);

typedef struct XrGraphicsRequirementsVulkanKHR {
    XrStructureType type;
    void* next;
    XrVersion minApiVersionSupported;
    XrVersion maxApiVersionSupported;
} XrGraphicsRequirementsVulkanKHR;

typedef struct XrSwapchainImageVulkanKHR {
    XrStructureType type;
    void* next;
    VkImage image;
} XrSwapchainImageVulkanKHR;

typedef struct XrGraphicsBindingVulkanKHR {
    XrStructureType type;
    const void* next;
    VkInstance instance;
    VkPhysicalDevice physicalDevice;
    VkDevice device;
    uint32_t queueFamilyIndex;
    uint32_t queueIndex;
} XrGraphicsBindingVulkanKHR;
```

Session Control [9.2]

```
XrResult xrBeginSession(XrSession session,
    const XrSessionBeginInfo* beginInfo);

typedef struct XrSessionBeginInfo {
    XrStructureType type;
    const void* next;
    XrViewConfigurationType primaryViewConfigurationType;
} XrSessionBeginInfo;

XrResult xrEndSession(XrSession session);

XrResult xrRequestExitSession(XrSession session);
```

Session States [9.3]

```
typedef struct XrEventDataSessionStateChanged {
    XrStructureType type;
    const void* next;
    XrSession session; XrSessionState state;
    XrTime time;
} XrEventDataSessionStateChanged;
```

state: XR_SESSION_STATE_*X* where *X* may be: UNKNOWN, IDLE, READY, SYNCHRONIZED, VISIBLE, FOCUSED, STOPPING, LOSS_PENDING, EXITING

Compositing

Compositing [10.5]

Composition layers are submitted by the application via the **xrEndFrame** call. All composition layers to be drawn must be submitted with every **xrEndFrame** 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 layerFlags;
    XrSpace space;
} XrCompositionLayerBaseHeader;
```

layerFlags: A bitwise OR of XR_COMPOSITION_LAYER_*X*_BIT where *X* 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 {
    XrStructureType type;  const void*  next;
    XrCompositionLayerFlags layerFlags;
    XrSpace space;
    uint32_t viewCount;
    const XrCompositionLayerProjectionView* views;
} XrCompositionLayerProjection;

typedef struct XrCompositionLayerProjectionView {
    XrStructureType type;
    const void* next;
    XrPosef pose;
    XrFovf fov;
    XrSwapchainSubImage subImage;
} XrCompositionLayerProjectionView;
```

XR_KHR_composition_layer_cube [12.25]

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

```
typedef struct XrCompositionLayerCubeKHR {
    XrStructureType type;  const void* next;
    XrCompositionLayerFlags layerFlags;
    XrSpace space;
    XrEyeVisibility eyeVisibility;
    XrSwapchain swapchain;
    uint32_t imageArrayIndex;
    XrQuaternionf orientation;
} XrCompositionLayerCubeKHR;
```

XR_KHR_composition_layer_cylinder [12.6]

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.

```
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;
```

XR_KHR_composition_layer_equirect [13.6]

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.

```
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;

typedef
    struct XrCompositionLayerColorModulationInfoKHR {
    XrStructureType type;
    const void* next;
    XrColor4f colorScale;
    XrColor4f colorOffset;
} XrCompositionLayerColorModulationInfoKHR;

typedef struct XrCompositionLayerDepthInfoKHR {
    XrStructureType type;
    const void* next;
    XrSwapchainSubImage subImage;
    float minDepth;
    float maxDepth;
    float nearZ;
    float farZ;
} XrCompositionLayerDepthInfoKHR;

typedef struct XrCompositionLayerQuad {
    XrStructureType type;
    const void* next;
    XrCompositionLayerFlags layerFlags;
    XrSpace space;
    XrEyeVisibility eyeVisibility;
    XrSwapchainSubImage subImage;
    XrPosef pose;
    XrExtent2Df size;
} XrCompositionLayerQuad;
```

eyeVisibility: XR_EYE_VISIBILITY_*X* where *X* may be: BOTH, LEFT, RIGHT

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;
    char actionSetName[XR_MAX_ACTION_SET_NAME_SIZE];
    char localizedActionSetName[
        XR_MAX_LOCALIZED_ACTION_SET_NAME_SIZE];
    uint32_t priority;
} 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;
    char actionName[XR_MAX_ACTION_NAME_SIZE];
    XrActionType actionType;
    uint32_t countSubactionPaths;
    const XrPath* subactionPaths;
    char localizedActionName[
        XR_MAX_LOCALIZED_ACTION_NAME_SIZE];
} XrActionCreateInfo;
```

actionType: XR_ACTION_TYPE_*X* where *X* may be: BOOLEAN_INPUT, FLOAT_INPUT, VECTOR2F_INPUT, POSE_INPUT, VIBRATION_OUTPUT

```
XrResult xrDestroyAction(XrAction action);
```

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 xrSuggestInteractionProfileBindings(
    XrInstance instance,
    const XrInteractionProfileSuggestedBinding*
        suggestedBindings);

typedef struct XrInteractionProfileSuggestedBinding {
    XrStructureType type;
    const void* next;
    XrPath interactionProfile;
    uint32_t countSuggestedBindings;
    const XrActionSuggestedBinding* suggestedBindings;
} XrInteractionProfileSuggestedBinding;

typedef struct XrActionSuggestedBinding {
    XrAction action;
    XrPath binding;
} XrActionSuggestedBinding;
```

Continued on next page >

Inputs and Haptics: Actions (continued)

```
typedef struct XrEventDataInteractionProfileChanged {
    XrStructureType type;
    const void* next;
    XrSession session;
} XrEventDataInteractionProfileChanged;
```

An action set becomes immutable when attathed to a session.

```
XrResult xrAttachSessionActionSets(XrSession session,
    const XrSessionActionSetsAttachInfo* attachInfo);

typedef struct XrSessionActionSetsAttachInfo {
    XrStructureType type;
    const void* next;
    uint32_t countActionSets;
    const XrActionSet* actionSets;
} XrSessionActionSetsAttachInfo;

XrResult xrGetCurrentInteractionProfile(XrSession session,
    XrPath topLevelUserPath,
    XrInteractionProfileInfo* interactionProfile);

typedef struct XrInteractionProfileInfo {
    XrStructureType type;
    const void* next;
    XrPath interactionProfile;
} XrInteractionProfileInfo;
```

Reading Input Action State [11.5]

```
typedef struct XrActionStateGetInfo {
    XrStructureType type;
    const void* next;
    XrAction action;
    XrPath subactionPath;
} XrActionStateGetInfo;

typedef struct XrHapticActionInfo {
    XrStructureType type;
    const void* next;
    XrAction action;
    XrPath subactionPath;
} XrHapticActionInfo;

XrResult xrGetActionStateBoolean(XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStateBoolean* state);

typedef struct XrActionStateBoolean {
    XrStructureType type;
    void* next;
    XrBool32 currentState;
    XrBool32 changedSinceLastSync;
    XrTime lastChangeTime;
    XrBool32 isActive;
} XrActionStateBoolean;

XrResult xrGetActionStateFloat(XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStateFloat* state);

typedef struct XrActionStateFloat {
    XrStructureType type;
    void* next;
    float currentState;
    XrBool32 changedSinceLastSync;
    XrTime lastChangeTime;
    XrBool32 isActive;
} XrActionStateFloat;

XrResult xrGetActionStateVector2f(XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStateVector2f* state);

typedef struct XrActionStateVector2f {
    XrStructureType type;
    void* next;
    XrVector2f currentState;
    XrBool32 changedSinceLastSync;
    XrTime lastChangeTime;
    XrBool32 isActive;
} XrActionStateVector2f;

XrResult xrGetActionStatePose(XrSession session,
    const XrActionStateGetInfo* getInfo,
    XrActionStatePose* state);

typedef struct XrActionStatePose {
    XrStructureType type;
    void* next;
    XrBool32 isActive;
} XrActionStatePose;
```

Output Actions and Haptics [11.6]

```
XrResult xrApplyHapticFeedback(XrSession session,
    const XrHapticActionInfo* hapticActionInfo,
    const XrHapticBaseHeader* hapticFeedback);

typedef struct XrHapticBaseHeader {
    XrStructureType type;
    const void* next;
} XrHapticBaseHeader;

typedef struct XrHapticVibration {
    XrStructureType type;
    const void* next;
    XrDuration duration;
    float frequency;
    float amplitude;
} XrHapticVibration;
```

duration: nanoseconds or XR_MIN_HAPTIC_DURATION
frequency: Hz or XR_FREQUENCY_UNSPECIFIED

```
XrResult xrStopHapticFeedback(XrSession session,
    const XrHapticActionInfo* hapticActionInfo);
```

Input Action State Synchronization [11.7]

```
XrResult xrSyncActions(XrSession session,
    const XrActionsSyncInfo* syncInfo);

typedef struct XrActionsSyncInfo {
    XrStructureType type;
    const void* next;
    uint32_t countActiveActionSets;
    const XrActiveActionSet* activeActionSets;
} XrActionsSyncInfo;

typedef struct XrActiveActionSet {
    XrActionSet actionSet;
    XrPath subactionPath;
} XrActiveActionSet;
```

Action Sources [11.8]

```
XrResult xrEnumerateBoundSourcesForAction(
    XrSession session,
    const XrBoundSourcesForActionEnumerateInfo*
      enumerateInfo,
    uint32_t sourceCapacityInput,
    uint32_t* sourceCountOutput,
    XrPath* sources);

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

XrResult xrGetInputSourceLocalizedName(
    XrSession session,
    const XrInputSourceLocalizedNameGetInfo* getInfo,
    uint32_t bufferCapacityInput,
    uint32_t* bufferCountOutput,
    char* buffer);

typedef struct XrInputSourceLocalizedNameGetInfo {
    XrStructureType type;
    const void* next;
    XrPath sourcePath;
    XrInputSourceLocalizedNameFlags whichComponents;
} XrInputSourceLocalizedNameGetInfo;
```

whichComponents: A bitwise OR of XR_INPUT_SOURCE_LOCALIZED_NAME_*X*_BIT where *X* may be: USER_PATH, INTERACTION_PROFILE, COMPONENT

Extensions [12]

Extension naming convention [2.6]

XR_KHR_*	Khronos-created extensions supported by multiple vendors
XR_EXT_*	extensions supported by multiple vendors, possibly IP-restricted

XR_KHR_convert_timespec_time [12.9]

Enabling this extension makes the following available.

```
XrResult xrConvertTimespecTimeToTimeKHR(
    XrInstance instance,
    const struct timespec* timespecTime, XrTime* time);

XrResult xrConvertTimeToTimespecTimeKHR(
    XrInstance instance, XrTime time,
    struct timespec* timespecTime);
```

XR_KHR_D3D11_enable [12.11]

Support the D3D 11 graphics API in an OpenXR runtime.

```
XrResult xrGetD3D11GraphicsRequirementsKHR(
    XrInstance instance, XrSystemId systemId,
    XrGraphicsRequirementsD3D11KHR*
    graphicsRequirements);

typedef struct XrGraphicsBindingD3D11KHR {
    XrStructureType type;
    const void* next;
    ID3D11Device* device;
} XrGraphicsBindingD3D11KHR;

typedef struct XrSwapchainImageD3D11KHR {
    XrStructureType type;
    void* next;
    ID3D11Texture2D* texture;
} XrSwapchainImageD3D11KHR;

typedef struct XrGraphicsRequirementsD3D11KHR {
    XrStructureType type;
    void* next;
    LUID adapterLuid;
    D3D_FEATURE_LEVEL minFeatureLevel;
} XrGraphicsRequirementsD3D11KHR;

XrResult xrDestroySession(XrSession session);
```

XR_KHR_D3D12_enable [12.12]

Support the D3D 12 graphics API in an OpenXR runtime.

```
XrResult xrGetD3D12GraphicsRequirementsKHR(
    XrInstance instance, XrSystemId systemId,
    XrGraphicsRequirementsD3D12KHR*
      graphicsRequirements);

typedef struct XrGraphicsBindingD3D12KHR {
    XrStructureType type;
    const void* next;
    ID3D12Device* device;
    ID3D12CommandQueue* queue;
} XrGraphicsBindingD3D12KHR;

typedef struct XrSwapchainImageD3D12KHR {
    XrStructureType type;
    void* next;
    ID3D12Resource* texture;
} XrSwapchainImageD3D12KHR;

typedef struct XrGraphicsRequirementsD3D12KHR {
    XrStructureType type;
    void* next;
    LUID adapterLuid;
    D3D_FEATURE_LEVEL minFeatureLevel;
} XrGraphicsRequirementsD3D12KHR;
```

XR_KHR opengl_enable [12.14]

Support theOpenGL graphics API in an OpenXR runtime.

```
typedef struct XrGraphicsBindingOpenGLWin32KHR {
    XrStructureType type;
    const void* next;
    HDC hDC; HGLRC hGLRC;
} XrGraphicsBindingOpenGLWin32KHR;

typedef struct XrGraphicsBindingOpenGLXlibKHR {
    XrStructureType type;
    const void* XR_MAY_ALIAS next;
    Display* xDisplay;
    uint32_t visualid;
    GLXFBConfig glxFBConfig;
    GLXDrawable glxDrawable;
    GLXContext glxContext;
} XrGraphicsBindingOpenGLXlibKHR;

typedef struct XrGraphicsBindingOpenGLXcbKHR {
    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;
} XrGraphicsBindingOpenGLXcbKHR;

typedef struct XrGraphicsBindingOpenGLWaylandKHR {
    XrStructureType type;
    const void* next;
    struct wl_display* display;
} XrGraphicsBindingOpenGLWaylandKHR;
```

Continued on next page >

Extensions (continued)

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

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

```
typedef struct XrGraphicsRequirementsOpenGLKHR {
    XrStructureType type;
    void* next;
    XrVersion minApiVersionSupported;
    XrVersion maxApiVersionSupported;
} XrGraphicsRequirementsOpenGLKHR;
```

XR_KHR_opengl_es_enable [12.15]

Support the OpenGL ES graphics API in an OpenXR runtime.

```
typedef struct XrGraphicsBindingOpenGLESAndroidKHR {
    XrStructureType type;
    const void* next;
    EGLDisplay display;
    EGLConfig config;  EGLContext context;
} XrGraphicsBindingOpenGLESAndroidKHR;
```

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

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

```
typedef struct XrGraphicsRequirementsOpenGLESKHR {
    XrStructureType type;
    void* next;
    XrVersion minApiVersionSupported;
    XrVersion maxApiVersionSupported;
} XrGraphicsRequirementsOpenGLESKHR;
```

XR_KHR_visibility_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

```
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;
```

```
typedef struct XrEventDataVisibilityMaskChangedKHR {
    XrStructureType type;
    const void* next;
    XrSession session;
    XrViewConfigurationType viewConfigurationType;
    uint32_t viewIndex;
} XrEventDataVisibilityMaskChangedKHR;
```

XR_KHR_vulkan_enable [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 bufferCapacityInput,
    uint32_t* bufferCountOutput, char* buffer);
```

```
XrResult xrGetVulkanDeviceExtensionsKHR(
    XrInstance instance, XrSystemId systemId,
    uint32_t bufferCapacityInput,
    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 viewFormatCount;
    const VkFormat* viewFormats;
} XrVulkanSwapchainFormatListCreateInfoKHR;
```

XR_KHR_win32_convert_performance_counter_time [12.19]

This extension enables the following:

```
XrResultxrConvertWin32PerformanceCounterToTimeKHR(
    XrInstance instance,
    const LARGE_INTEGER* performanceCounter,
    XrTime* time);
```

```
XrResultxrConvertTimeToWin32PerformanceCounterKHR(
    XrInstance instance, XrTime time,
    LARGE_INTEGER* performanceCounter);
```

Notes

Learn more about OpenXR

OpenXR is maintained by the Khronos Group, a worldwide consortium of organizations that work to create and maintain key standards used across many industries. Visit Khronos online for resources to help you use and master OpenXR:

Main OpenXR Resource Page:	khronos.org/openxr/
OpenXR Registry:	khronos.org/registry/openxr/
Forums:	forums.khronos.org/
Slack:	khr.io/slack
Courses:	khronos.org/developers/training/
Videos & Presentations:	khr.io/library/
Khronos Events:	khronos.org/news/events/
Khronos Blog:	khronos.org/blog/
Reference Guides:	khr.io/refguides/
Khronos Books:	khronos.org/developers/books/
Khronos Merchandise:	khronos.org/store/

@thekhronosgroup khronos.org