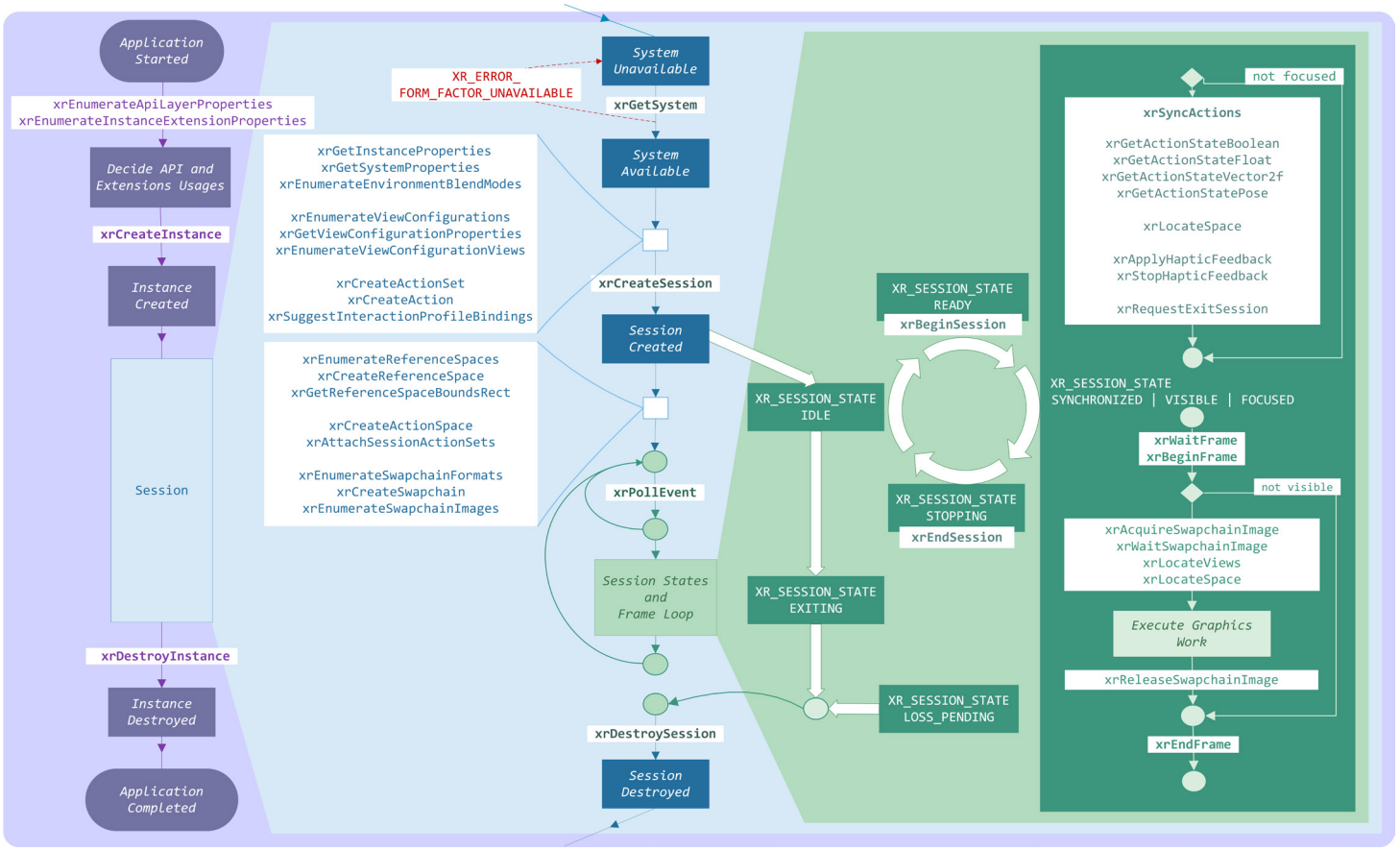
OpenXR™ 是一个是一个跨平台的API，借助它计算机可以生成一个虚拟与现实相结合的环境，并允许用户与之进行交互，它包含虚拟现实、增强现实、及混合现实相关等技术。它是应用与XR运行时的交互接口，可以处理处理帧合成、外设管理等工作。

规范和附加资源详见 khronos.org/openxr

具有颜色编码的名称: function names and structure names.
[n.n.n] 表示 OpenXR 1.0 规范中的部分和文本。
O 表示属于扩展的内容。

OpenXR API 概述

典型 OpenXR 应用程序，包括函数顺序调用、对象创建、会话状态更改和渲染循环。



OpenXR Action(行为) 系统概念 [11.1]

创建 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"

设置交互配置绑定

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", ... }

行为(action) 的同步与获取

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

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

OpenXR 分离应用行为(action)与输入，例如将输入 Trigger、Thumbstck、Button 等事件与移动、跳转和传送等行为(action)分离。行为(action)被分组到对应的应用程序定义行为集(action set)被上下文（菜单、游戏玩法等）使用。 这简化和预留了对不同输入设备的支持并最大限度地提高用户的可访问性。

交互配置以实体设备路径按顺序标识按钮和其他输入源的集合，以允许应用程序和OpenXR Runtime对行为(action)与输入进行映射。
OpenXR Runtime 根据应用程序的建议和其他特定runtime源将行为(action)与输入设备进行绑定。这允许开发人员使用和测试定制硬件，同时可以使用OpenXR Runtime支持的其他硬件。

xrSyncActions的输入为行为集(active set)，执行即要求运行时更新动作状态。大多数设备输入都可以通过 xrGetActonState* 函数获得。"action space"和xrLocateSpace用于追踪对象位置姿势，例如用于空间位置参考。

OpenXR 基础知识

指针链遍历(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;
```

缓冲区大小(Buffer size parameters)[2.11]

一些形式参数为输入/输出buffer的函数具有以下形式：

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

两次调用(Two-call idiom for buffer size parameters)

初次调用xrFunction()使用uint32_t* elementCountOutput，elements = NULL, elementCapacityInput = 0;其中，elementCountOutput 用来接收元素数量。然后分配足够空间使用elements = [缓冲区]作为参数再次调用。

版本宏与控制宏(Macros for version and header control)

版本号(Version numbers) [2.1, Appendix]

typedef uint64_t XrVersion;
版本号以 64 位编码，如下所示：

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

版本宏(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)
```

图形API控制宏 [Appendix]

编译时符号(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

操作(视窗)系统控制宏 [Appendix]

编译时符号(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	XR Wayland
_USE_PLATFORM_ANDROID	Android Native

线程(Threading behavior) [2.3]

OpenXR 函数通常支持从多个线程调用，但有一些例外：

- 将被销毁函数销毁的句柄参数和任何子句柄必须在外部同步。
  - 实例参数(instalce)和任何子句柄 xrDestroyInstance
  - 会话(session)参数和任何子句柄 xrDestroySession
  - 空间参数(space)和任何子句柄 xrDestroySpace
  - 交换链(swapchain)参数和任何子句柄 xrDestroySwapchain
  - 行为集(actionSet)参数和任何子句柄 xrDestroyActionSet
  - 行为(action)参数和任何子句柄 xrDestroyAction
- 对给定 XrSession 的 xrWaitFrame 调用必须是外部同步。

XR_KHR_android_thread_settings [12.3]

如果启用此扩展,应允许应用程序指定Android 线程类型。

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

threadType: XR_ANDROID_THREAD_TYPE_*X*_KHR
其中 *X* 可能是:
APPLICATION_MAIN, APPLICATION_WORKER,
RENDERER_MAIN, RENDERER_WORKER

时间(Time)

XrTime [2.12.1]

一个64位整数，表示一个time相对于一个运行时依赖(runtime-dependent)时间。所有同时应用程序使用相同的时间。在android中与std::chrono::steady_clock::now().time_since_epoch().count()一致。

XrDuration [2.13]

表示持续时间的64位有符号整数；两个XrTime值之间的差异。

特殊值(Special values):

```
#define XR_NO_DURATION 0
#define XR_INFINITE_DURATION 0x7fffffffffffffLL
```

数据类型

颜色 [2.14]

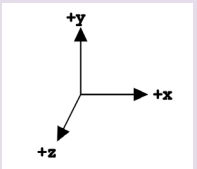
颜色分量是线性的范围为 0.0..1.0，而不是 sRGB，也没有做 alpha-premultplied。

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

坐标系(Coordinate system) [2.15]

OpenXR 使用带有 x、y 和 z 轴的笛卡尔右手坐标系。

可以使用以下具有以下成员的结构类型来表示点和方向：



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;
```

返回值(XrResult return codes) [2.8]

API命令返回XrResult类型的值。负值是错误代码，而非负(≥0)是成功代码。

成功代码(Success codes)

XR_SUCCESS
XR_SESSION_LOSS_PENDING
XR_SESSION_NOT_FOCUSED
XR_SPACE_BOUNDS_UNAVAILABLE
XR_TIMEOUT_EXPIRED
XR_EVENT_UNAVAILABLE
XR_FRAME_DISCARDED

错误代码(Error codes)

XR_ERROR_*X* 其中 *X* 可能是:

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
这个扩展被启用的时候会存在下面错误：
XR_ERROR_ANDROID_THREAD_SETTINGS_ID_INVALID_KHR
XR_ERROR_ANDROID_THREAD_SETTINGS_FAILURE_KHR

工具宏 [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.

生命周期(Instance lifecycle)

API layers and extensions [2.7, 4.1]

API层插在应用程序和 runtme 之间，通过 Hook API 方式进行日志记录、调试、验证等。扩展可以公开新功能或修改现有功能的行为。在XrInstance创建时选择扩展和API层。要启用一个API层，请将其名称添加到XrInstanceCreateInfo的enabledApiLayerNames中。要启用扩展，请将其名称添加到XrInstanceCreateInfo的enabledExtensions中。

```
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]

调用 xrCreateInstance 以获取 XrInstance 句柄。
Instance 管理应用程序和 OpenXR Runtime 之间的接口。

```
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 一定为 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]

此扩展支持以下功能：

```
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]

接收此结构可预测在lossTime处的会话(session)丢失。
应用程序应调用xrDestroyInstance并释放实例资源。

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

```
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;
```

类型转字符串(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]);
```

常用类型(Common types)

偏移量、范围和面积(Offsets, extents, and areas) [2.16]

物理角度，成员以米为单位表示偏移。(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;
```

物理角度，成员指定一个以米为单位的矩形区域。

```
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)

物理角度，成员指定一个以米为单位的矩形区域。

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

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

(Members specify a rectangular area in meters if physical)

FOV angles [2.17]

以弧度为单位，从 -π/2 到 π/2。

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

Boolean type [2.19]

只有 XR_TRUE 或 XR_FALSE可用.

```
typedef uint32_t XrBool32;
```

Event polling [2.20.1]

应用程序应分配 XrEventDataBufer 类型的事件队列并定期调用 xrPollEvent。如果事件队列溢出，xrPollEvent 将返回 XrEventDataEventsLost 事件

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

语义路径和路径树(Semantic Paths and Path Tree)

路径名和XrPath [6.1, 6.2]

路径名称字符串只能包含小写字母 a-z、数字 0-9、连字符、下划线、句点或正斜杠。

XrPath 是原子的不可分的，它在单个实例的上下文中将应用程序与唯一路径连接起来。 由于 XrPath 只是良好格式的路径字符串的简写，它们没有明确的生命周期。

路径转字符串(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);
```

保留路径(Reserved paths) [6.3.1]

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

输入输出子路径(Input/output subpaths) [6.3.2-3]

输入路径具有以下形式:

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

For extensions, the form is:

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

触觉等设备的路径名遵循以下形式：

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

Continued on next page >

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;
```

语义路径和路径树 (Semantic Paths and Path Tree continue)

标识符(identifier)标准值

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

dpad_*X*其中*X*可以是: up, down, left, right

diamond_*X*其中*X*可以是: up, down, left, right

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

volume_up, volume_down, mute_mic, play_pause, menu

标识符(identifier)标准姿势

grip aim

标准位置(location)

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 (震动)

Interaction profile paths [6.4]

交互配置标识按钮和其他输入源的集合，其形式为：

/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;

空间位置(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*);

运行时应支持R8G8B8A8和R8G8B8A8 sRGB 格式。使用基于OpenGL的图形API，纹理格式对应于OpenGL内部格式。使用基于Direct3D的图形API，**xrEnumerateSwapchainFormats** 永远不会返回无类型格式。 仅返回具体格式，或者可以由应用程序指定用于创建交换链。

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:0 或跟下面按位或
XR_SWAPCHAIN_CREATE_PROTECTED_CONTENT_BIT,
XR_SWAPCHAIN_CREATE_STATIC_IMAGE_BIT
usageFlags:0 或跟下面按位或
XR_SWAPCHAIN_USAGE_*X*_BIT where *X* may be:
COLOR_ATTACHMENT,
DEPTH_STENCIL_ATTACHMENT, UNORDERED_ACCESS,
TRANSFER_SRC, TRANSFER_DST, SAMPLED,
MUTABLE_FORMAT
参数*sampleCount, width,height, mipcount:*不能为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 这里X 可能是 :
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]

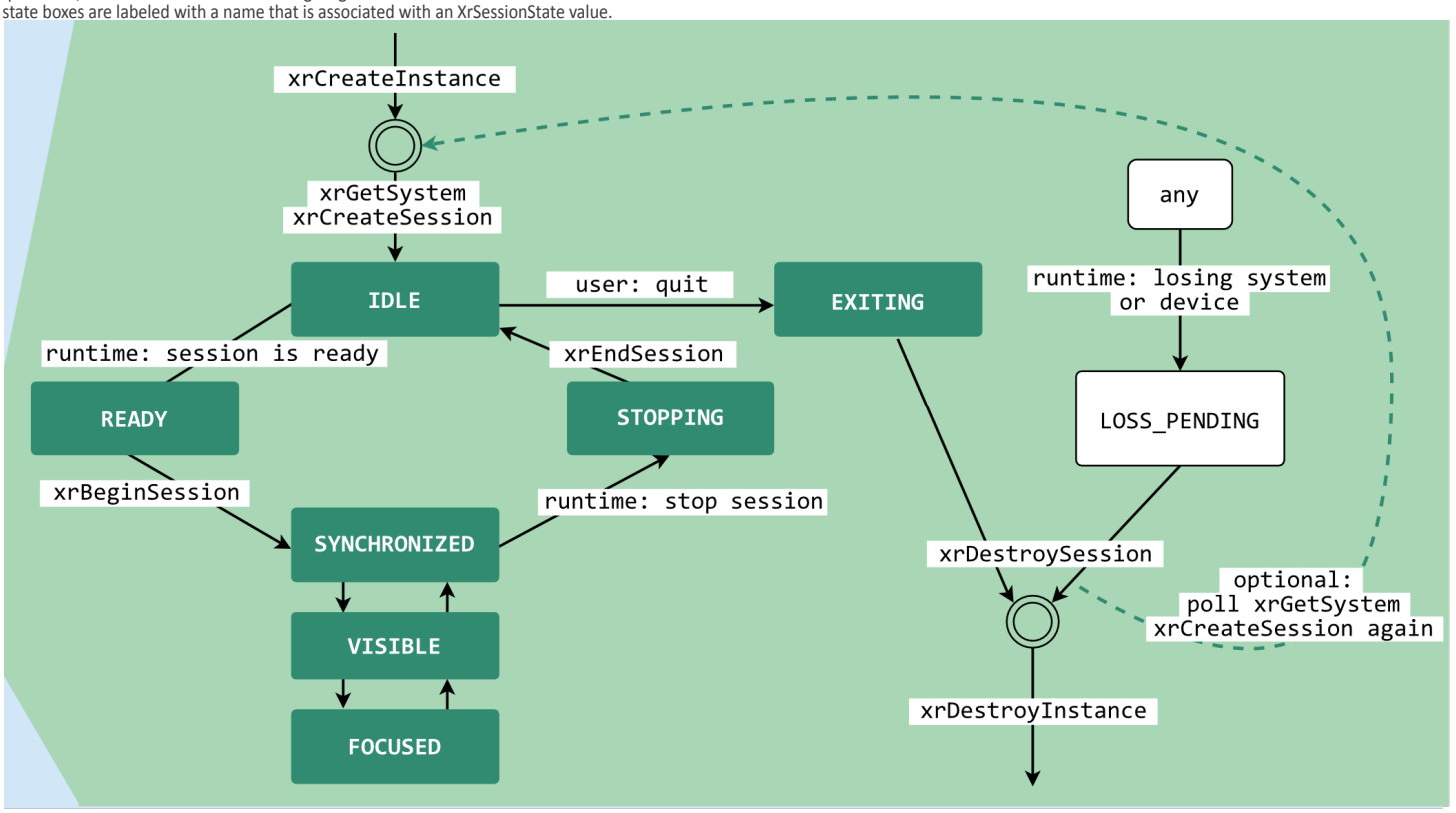
XrResult **xrEnumerateEnvironmentBlendModes**(
XrInstance *instance*, XrSystemId *systemId*,
XrViewConfigurationType *viewConfigurationType*,
uint32_t *environmentBlendModeCapacityInput*,
uint32_t* *environmentBlendModeCountOutput*,
XrEnvironmentBlendMode* *environmentBlendModes*);

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]

XRSession基于应用程序请求，运行时操作量和用户操作，通过多个状态进行。下图显示了会话状态计算机。状态框标有与XRSession State值相关的名称。

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

行为(action):输入输出(震动触感) Input and Haptics: Actions

行为(action)应当在初始化的时候创建一次，之后被用在获取输入状态，创建行为空间(action space)，和发送震动触感事件。

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]

应用程序应当向Runtime提供默认绑定，以便输入数据可以适当地映射到应用程序行为(action)。下面系统建议的绑定仅为运行时的提示。

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 >

行为(action):输入输出(震动触感)

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

当一个行为(action)附加到一个会话(session)的时候，它将变成不可变的(immutable)。

```
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;
```

获取输入状态 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: 跟*XR_INPUT_SOURCE_LOCALIZED_NAME_X_BIT* 按位或 ，其中*X* 可以是: USER_PATH, INTERACTION_PROFILE, COMPONENT

扩展 Extensions [12]

扩展名约定 Extension naming convention [2.6]

XR_KHR_*	由Khronos创建的扩展，由多个供应商支持。
XR_EXT_*	由供应商支持的扩展名，可能受到IP限制。

XR_KHR_convert_timespec_time [12.9]

启用该扩展后，下面功能开放可用。

```
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]

此扩展程序可以启用以下内容：

```
XrResult xrGetVisibilityMaskKHR(XrSession session,
    XrViewConfigurationType viewConfigurationType,
    uint32_t viewIndex,
    XrVisibilityMaskTypeKHR visibilityMaskType,
    XrVisibilityMaskKHR* visibilityMask);
```

visibilityMask:
XR_VISIBILITY_MASK_TYPE_X_KHR 其中 X 可以是:
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]

支持Additon中的OpenXr Runtme中的Vulkan Graphics API, 见本参考指南第6页。

```
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]

此扩展程序可以启用以下内容：

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

XR_KHR_win32_convert_performance_counter_time [12.19]

此扩展程序可以启用以下内容：

```
XrResult xrConvertWin32PerformanceCounterToTimeKHR(
    XrInstance instance,
    const LARGE_INTEGER* performanceCounter,
    XrTime* time);

XrResult xrConvertTimeToWin32PerformanceCounterKHR(
    XrInstance instance, XrTime time,
    LARGE_INTEGER* performanceCounter);
```

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

Notes

OpenXR is a trademark of the Khronos Group. The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics, and dynamic media on a wide variety of platforms and devices.

See www.khronos.org to learn more about the Khronos Group.
See www.khronos.org/openxr to learn more about OpenXR.