联　　盟　　标　　准

GSXR 互通应用层接口与基础能力规范
GSXR-A-01-04

2021－09－10 发布

GSXR 工作组 发布

版权声明

前 言

本标准由中国移动通信集团终端有限公司起草，联合中国信息通信研究院、中国电信集团公司、中国联合网络通信集团有限公司、咪咕文化科技有限公司、宏达通讯有限公司（HTC VIVE）、北京凌宇智控科技有限公司共同撰写。在编写过程中得到了中国移动（成都）产业研究院、上海影创信息科技有限公司、华为技术有限公司、深圳创维新世界科技有限公司、海信聚好看科技股份有限公司、北京搜狗科技发展有限公司提出的宝贵建议。感谢青岛小鸟看看科技有限公司、中科创达软件股份有限公司、亮风台(上海)信息科技有限公司、北京爱奇艺科技有限公司、虚拟现实产业推进会（VRPC）等广大伙伴提供的专业支持。

本标准起草单位：

中国移动通信集团有限公司

本标准主要起草人（排名不分先后）：

倪茂、刘峰、骆志伟、王剑

编写组成员（排名不分先后）：

中国信息通信研究院：陈曦、宫政、胡可臻

中国电信集团公司：武娟、刘晓军

中国联合网络通信集团有限公司：蔡庆宇、李蓉

咪咕文化科技有限公司：刘永清、朱佳伟、朱奇

宏达通讯有限公司（HTC VIVE）：华晓枫、余思杰

北京凌宇智控科技有限公司：张佳宁、陈曦

VRPC秘书处：全一

引 言

GSXR（General Standard for XR)互通接口规范，提供了XR标准API（Application Programming Interface）接口定义与功能描述,包括虚拟现实(VR, Virtual Reality),增强现实(AR, Augmented Reality）和混合现实（MR, Mixed Reality）相关应用的标准接口。

XR应用开发者可以调用格式一致的XR功能函数开发XR应用，无需再为不同Runtime接口进行适配，只需专注于XR内容开发，且开发出的GSXR应用可运行于各个支持GSXR标准的XR设备上。目前，此规范只适用于安卓系统。

目 录

1. 概述

本文将提供 GSXR 互通接口的详细定义及基础能力的实现方式，为 GSXR 应用层接口定义完整的互通接口规范（以下简称“规范”）。

GSXR 互通接口是一系列为 XR 功能制定的函数集，依据 XR 功能区分类别，定义各函数相应的输入输出参数及返回值。本规范将根据 XR 互通接口函数，描述详细的接口定义，以及 XR 应用程序、XR Runtime 实际操作及实现细项。

2. 适用范围

XR 应用开发

开发者根据应用所需，执行对应的 XR 函数操作，例如获取头显或手柄设备状态，获取设备跟踪数据和交互操作、以及提交渲染帧等。

XR Runtime

XR Runtime 必须根据互通接口的定义实现相应函数，同时将每个函数功能适配到对应的 XR 设备，正确地反映出应用需求的操作行为。

3. 术语及缩略语

下列术语、缩略语适用于本规范：

表3-1　术语及缩略语列表

缩略语	全名
GSXR	General Standard for XR
规范	GSXR 互通接口规范
XR 应用	GSXR 应用程序
XR Runtime	GSXR Runtime
XR 设备	GSXR 设备
VR	Virtual Reality 虚拟现实
AR	Augmented Reality 增强现实
MR	Mixed Reality 混合现实
API (接口)	Application Programming Interface 应用程序接口
HMD (头显)	Head-Mounted Display 头戴式显示器
IPD	Inter Pupillary Distance 双眼瞳距
DOF	Degree Of Freedom 空间自由度
V-Sync	Vertical Synchronization 垂直同步
CPU	Central Processing Unit 中央处理单元
GPU	Graphics Processing Unit 图形处理单元

4. 主线流程

GSXR 应用程序对于互通接口的操作，从创建 Runtime 实例开始，该实例是 Runtime 的生成基础,所有 Runtime 之后的后续操作都将依据该实例进行。随后，Runtime 进行 XR 设备连线的接入，赋予设备正确状态及组态，确认跟踪、输入输出、图形及显示系统正常工作。随后创建渲染器实例及纹理队列并开始渲染循环，在渲染循环中，XR 应用程序等待 Runtime 返回正确的 V-Sync 信号，从而开

始获取 Runtime 的事件、位姿、输入数据，并获取纹理队列中的图形缓冲进行视图渲染，随后提交渲染帧予 Runtime 进行后续 XR 的图形算法处理并输出显示屏。

下图描述 XR 互通接口的基础主线流程。

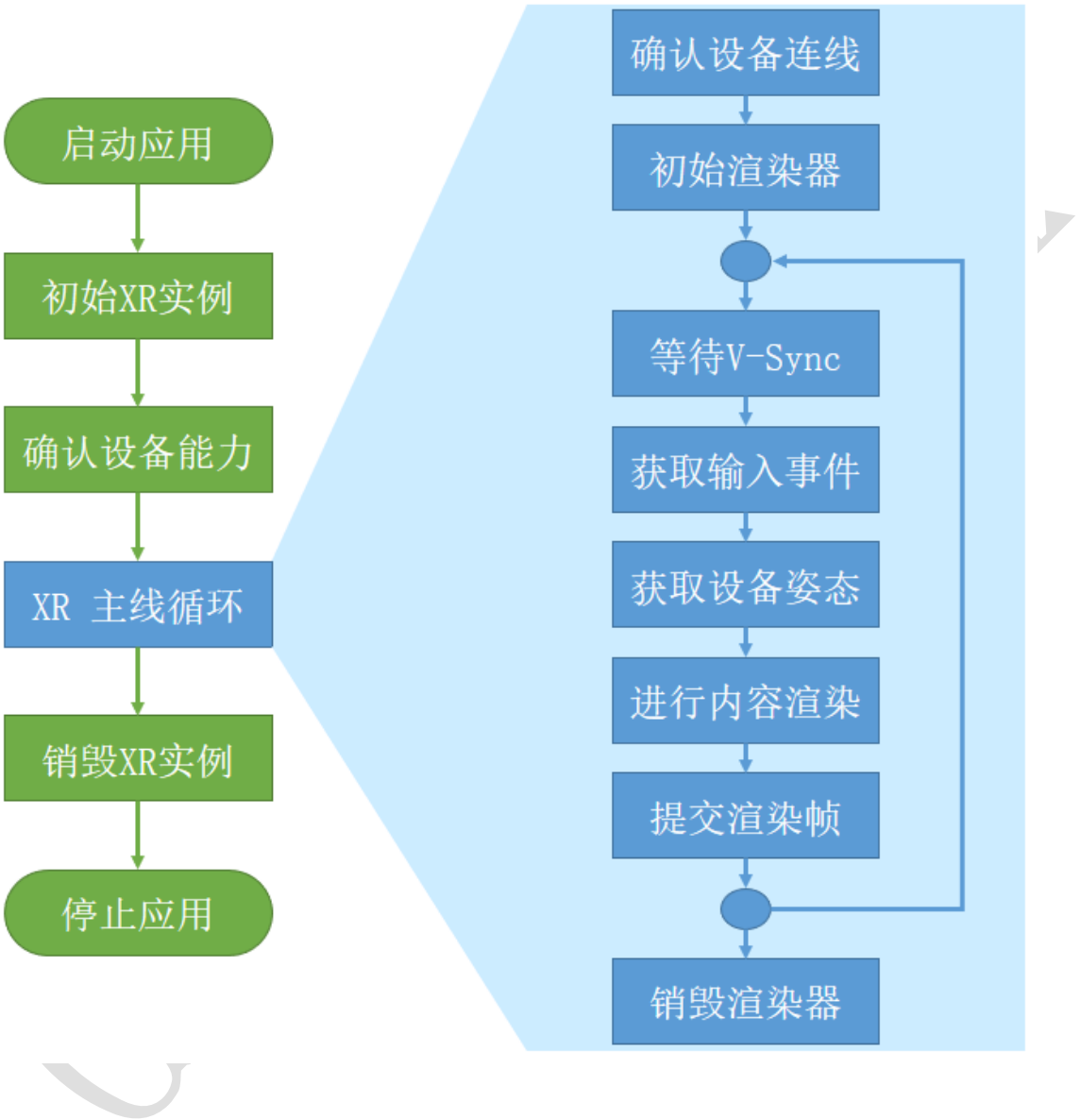


图4-1 XR互通接口主线流程

5. 通用基础类型

本章节描述 GSXR 互通接口的基本类型定义，后续章节针对 XR 功能函数的描

述都是以本章节作为通用原则，并在此基础上延伸其功能定义。

5.1. 版本号

版本号 GSXR_Version 为 GSXR 互通接口描述版本信息的标准类型，包括 XR API 版本号,XR Runtime 版本号都应使用 GSXR_Version 定义的版本号描述格式。

```
typedef uint64_t GSXR_Version;
```

GSXR_Version 为 64 位整数，区分三段：

- **补丁版本号（bit 31-0）**

  补丁版本号不同表示：函数定义做了小部分修改，通常是为了修复错误，且有可能小幅影响现有行为，也可能添加额外的 API 接口。修改补丁版本号不可影响两个版本之间的向后及向前兼容。

- **次版本号（bit 47-32）**

  次版本号不同表示：添加了某些新功能，这通常会在头文件中创建新接口，也可能包含行为更改和错误修复。函数可以在新的次版本中被宣告废弃，但不可删除。修改新的次版本号时，补丁版本号将重置为 0，再依补丁版本号规则进行后续维护。修改次版本号不可影响向后兼容，但可能影响向前兼容。

- **主版本号（bit 63-48）**

  主版本号不同表示：版本之间的函数进行了大量更改，可能包括新功能的引入，操作行为的更改，废弃函数的删除，原有函数的修改或替换，因此

极有可能破坏其向后及向前兼容。主版本号之间的差异通常需要对应用代码进行大部修改，以适配新版本的定义，使其正常运行。

GSXR 提供以下宏定义，方便获取规范版本号：

[版本号宏定义]
GSXR_CURRENT_API_VERSION：获取当前版本号
GSXR_VERSION_MAJOR (version)：从 version 中取出主版本号
GSXR_VERSION_MINOR (version)：从 version 中取出次版本号
GSXR_VERSION_PATCH (version)：从 version 中取出补丁版本号

例如：当 GSXR 规范版本号为 1.0.0 时，GSXR_Version 为 0000 0000 0000 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000。若之后小幅度修改了不影响向后及向前兼容的 API 行为或新增了 API 接口，可更新一个补丁版本 1.0.1，GSXR_Version 为 0000 0000 0000 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001。若之后废弃了某 API 接口，可宣告其为废弃函数，并更新一个次版本 1.1.0，GSXR_Version 为 0000 0000 0000 0001 0000 0000 0000 0001 0000 0000 0000 0000 0000 0000 0000 0000，只有此版本仍必须维持向后兼容，但可能影响向前兼容。

5.2. 布尔值

GSXR_Bool32 为 GSXR 互通接口描述布尔值的 32 位整数，GSXR_TRUE 为真值，GSXR_FALSE 为假值。

```
typedef uint32_t GSXR_Bool32;
```

5.3. 标志值

GSXR_Flags64 为 GSXR 互通接口描述位掩码的 64 位整数，通常用于特定功能的枚举类型，具备多项成员。每一成员以一位位元表示，按需求选用成员，选用其代表位元进行 OR 运算后的位掩码，即表示该功能选项的有效组合。

```
typedef uint64_t GSXR_Flags64;
```

5.4. 字符串

GSXR 互通接口使用的字符串(char)全部都是 UTF-8 编码并以空字符（‘\0’）结尾。GSXR 函数的输入输出字符串变数通常以 128 为最大长度。

```
#define GSXR_COMMON_STRING_MAX_SIZE 128
```

5.5. 时间戳

GSXR_Time 是一个 64 位的有符号整数，用以表示某个时序点的时间戳，以 naroseconds 为单位。时间戳必须是依循时序递增的常数值，不会因为系统区域时间调整而受影响。当 GSXR_Time 递增至超过 int64_t 所允许的最大正数时，XR Runtime 必须适时将时间戳归零，避免时间戳数值溢出造成非预期的错误行为。

```
typedef int64_t GSXR_Time;
```

5.6. 持续时间

GSXR_Duration 是一个 64 位的有符号整数，用以表示经过的某段时间区段，而不是某个特定的时序点。同样地，可表述为两个时间戳（GSXR_Time）相减的差值便是此二时序点的持续时间（GSXR_Duration）。

```
typedef int64_t GSXR_Duration;
```

GSXR_Duration 通常用于指示某项操作的持续时间，或是某项条件的等待时间。若对象具备时效性，可用 GSXR_NO_DURATION 代表立即结束操作或等待并返回。若对象为一必备条件或元素，可用 GSXR_INFINITE_DURATION 代表时间持续至条件发生时才返回。

```
#define GSXR_NO_DURATION                0
#define GSXR_INFINITE_DURATION          0x7fffffffffffffffLL
```

5.7. 句柄

GSXR 互通接口创建某特定功能的实例时，句柄作为此功能实例的代表，XR 应用程序调用功能函数时都需指定对应句柄为输入参数，用来识别操作的实例对象。

```
#define GSXR_DEFINE_HANDLE(object) typedef void* object;
```

句柄的生成必须由 XR Runtime 确定完成实例创建后返回，有效的句柄不可

为 GSXR_NULL_HANDLE。应用程序输入的句柄若为 GSXR_NULL_HANDLE 或不是由 XR Runtime 生成的句柄值，XR Runtime 全部将视为无效句柄。

```
#define GSXR_NULL_HANDLE 0
```

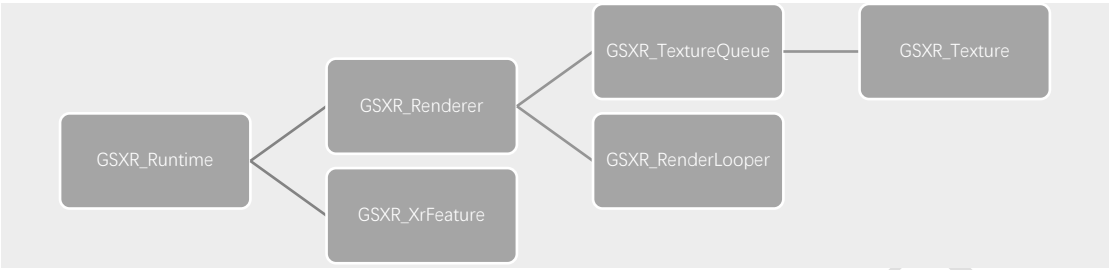
由于句柄的生成代表着某个实例的创建，其生命周期通常由一组成对的函数所管控（创建 Create*／销毁 Destroy*，开始 Start*／停止 Stop*，获得 Acquire*／释放 Release*），XR 应用程序必须在功能不使用时调用对应的结束函数，让 XR Runtime 更有效的管控实例成本并节省资源。

以下为当前定义的 GSXR 句柄类型，隶属该功能类型的函数全部需要输入 XR Runtime 生成的有效句柄进行识别，才可进行后续操作。若 XR Runtime 判定输入句柄为无效句柄，将返回 GSXR_Error_Handle_Invalid 错误码，XR 应用程序需重新检视句柄生命周期的有效性。

```
GSXR_DEFINE_HANDLE(GSXR_Runtime)          GSXR Runtime 句柄
GSXR_DEFINE_HANDLE(GSXR_Renderer)         渲染器句柄
GSXR_DEFINE_HANDLE(GSXR_TextureQueue)     纹理队列句柄
GSXR_DEFINE_HANDLE(GSXR_Texture)          纹理句柄
GSXR_DEFINE_HANDLE(GSXR_RenderLooper)     渲染循环句柄
GSXR_DEFINE_HANDLE(GSXR_XrFeature)        XR 特性句柄
```

句柄具有层次特性，子句柄必须依存于父句柄后创建，同样子句柄的生命周期也将依存父句柄而生。当父句柄被销毁时，即便应用程序尚未销毁子句柄，依

存于父句柄下的子句柄也连带遭遇销毁。句柄依存关系如下所示：



5.8. 标识号

在同一运行平台，对于某些允许接入多个设备的设备类型（如：显示屏），GSXR 互通接口定义了标识号识别设备对象。例如：创建渲染器时必须输入对应的显示标识号，用以识别输出的显示屏对象，而显示标识号只能从头显设备获取，手柄设备并无显示标识号。

```
#define GSXR_DEFINE_ATOM(object) typedef uint64_t object;
GSXR_DEFINE_ATOM(GSXR_DisplayId)        显示标识号
```

标识号的生成必须由 XR Runtime 根据平台及其设备管理规则配置后返回，有效的标识号不可为 0。由应用程序输入的句柄若为 0 或为不是由 XR Runtime 生成的标识号，XR Runtime 全部视为无效。

```
#define GSXR_NULL_DISPLAY_ID 0
```

5.9. 函数返回值

GSXR_Result 枚举各种可能发生的函数返回值，返回 0 或正值代表执行成功或处于某种预期内的正常状态，返回负值则代表执行失败或遭遇某种非预期的错误状态。

以下枚举当前定义的返回值，XR Runtime 必须根据函数执行结果返回相符的返回值，XR 应用程序必须根据函数调用后的返回值检视执行结果，并于应用程序中做出适合的对应处理。

[枚举] GSXR_Result（函数返回值）

名称	数值	描述
GSXR_Result_Success	0	函数调用成功
GSXR_Result_Event_Unavailable	1	事件队列中已无事件
GSXR_Result_Timeout_Expired	2	等待超时
GSXR_Result_Device_Connection_Pending	3	直至设备连线
GSXR_Error_Operation_Failed	-1	函数运行失败
GSXR_Error_API_Unsupported	-2	Runtime 未支持此 API
GSXR_Error_ApplicationType_Unsupported	-3	设备未支持此应用类型
GSXR_Error_Function_Unsupported	-4	设备未支持此功能
GSXR_Error_Feature_Unsupported	-5	设备未支持此特性
GSXR_Error_Device_Disconnected	-6	设备未连接
GSXR_Error_Handle_Invalid	-7	无效的输入句柄
GSXR_Error_Argument_Invalid	-8	无效的输入参数
GSXR_Error_Call_Flow_Invalid	-9	函数调用顺序错误

GSXR_Error_Out_Of_Memory	-10	内存溢出
GSXR_Error_Runtime_Unavailable	-11	Runtime 未处于正确工作状态
GSXR_Error_Renderer_Unavailable	-12	渲染器未处于正确工作状态
GSXR_Error_Feature_Unavailable	-13	特性未处于正确工作状态

5.10. 事件

XR Runtime 运行期间，Runtime 可能产生某些必需通知 XR 应用程序的状态信息，此时 XR 应用程序便通过轮询 XR Runtime 的事件队列获取事件。事件获取方法涉及事件队列轮询机制，将于章节 6.5 事件轮询详述。

5.11. 坐标系及姿态

GSXR 采用右手坐标系及常用右手定则(拇指指向轴向，其余四指指向旋转正方向)。

以下介绍 GSXR 对于二维矢量、三维矢量、四元数、位姿、及速度的结构体定义，GSXR 的空间坐标相关函数将大量使用到这些结构体。

[结构体] GSXR_Vector2f (二维矢量)

```
typedef struct GSXR_Vector2f {
    float    x;
    float    y;
} GSXR_Vector2f;
```

说明：

二维矢量结构体，用以表示二维空间坐标

成员：

x 为二维矢量 x 坐标

y 为二维矢量 y 坐标

备注：

若用于表示实际空间距离，单位为米

[结构体] GSXR_Vector3f（三维矢量）

```
typedef struct GSXR_Vector3f {
    float    x;
    float    y;
    float    z;
} GSXR_Vector3f;
```

说明：

三维矢量结构体，用以表示三维空间坐标

成员：

x 为三维矢量 x 坐标

y 为三维矢量 y 坐标

z 为三维矢量 z 坐标

备注：

若用于表示实际空间距离，单位为米

[结构体] GSXR_Quaternionf（四元数）

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

说明：

四元数结构体，用以表示旋转坐标

成员：

x 为四元数 x 坐标

y 为四元数 y 坐标

z 为四元数 z 坐标

w 为四元数 w 坐标

[结构体] GSXR_Posef (位姿)

```
typedef struct GSXR_Posef {
    GSXR_Vector3f         position;
    GSXR_Quaternionf      orientation;
} GSXR_Posef;
```

说明：

位姿结构体，用以表示位置及方向坐标

成员：

position 表示空间中的位置

orientation 表示空间中的方向

[结构体] GSXR_Velocityf (速度)

```
typedef struct GSXR_Velocityf {
    GSXR_Vector3f         linearVelocity;
    GSXR_Vector3f         angularVelocity;
} GSXR_Velocityf;
```

说明：

速度结构体，用以表示线速度及角速度

成员：

linearVelocity 表示线速度

angularVelocity 表示角速度

6. XR Runtime 函数说明

在 XR Runtime 开始运行前，XR 应用程序必须创建并初始化 XR Runtime 实例，才可以开始进行后续的 GSXR 函数操作。且 XR 应用程序在停用前同样必须销

毁 XR Runtime 实例，确保 XR Runtime 的生命周期可以被正确管理。

本章描述 GSXR 的基础函数，除详细说明 XR Runtime 实例的创建及销毁函数之外，也包含了返回值转换字符串、注册应用程序渲染函数等通用函数。而在 XR Runtime 实例创建成功之后，也可经由属性函数获取 XR Runtime 属性信息，且 XR 应用程序运行期间也必须周期性调用事件轮询函数获取即时事件，做出相应处理。

6.1. 返回值转换字符串

XR 应用程序调用函数后将返回 GSXR_Result，应用程序通常需要将返回值打印在日志中供排除错误或其他用途使用。GSXR 提供了将返回值转换为字符串的通用函数 GSXR_ResultToString 供 XR 应用程序使用。

[函数] GSXR_ResultToString (返回值转换字符串)

```
GSXR_Result GSXR_ResultToString (
    GSXR_Result            result,
    char                   string[GSXR_COMMON_STRING_MAX_SIZE]);
```

说明：

转换 GSXR_Result 返回值为字符串

参数：

[in] result GSXR_Result 返回值

[out] string 输出转换的字符串

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_API_Unsupported

GSXR_Error_Argument_Invalid

备注：

无

[基础能力规范] GSXR_ResultToString

[参数检验]

- `result` 必须是 GSXR_Result 定义的合法成员。
- `string` 必须是长度为 GSXR_COMMON_STRING_MAX_SIZE 的字符数组。

[函数实现]

- 将 result 转换成以 NULL（‘\0’）结尾的字符串，并存于 string 字符数组中输出。

[返回值]

成功

- `GSXR_Result_Success` 字符串转换成功。

失败

- `GSXR_Error_Operation_Failed` 字符串转换失败。
- `GSXR_Error_API_Unsupported` XR Runtime 未实现此 API。
- `GSXR_Error_Argument_Invalid` result 判定为不合法，或 string 数组长度不正确。

6.2. XR Runtime 的生命周期

XR 应用程序必须先完成 XR Runtime 的实例创建及初始，XR Runtime 才可以开始运行并提供 XR 应用程序必要的信息数据进行渲染，在 XR 应用程序离开前必须停止 XR Runtime 并销毁实例，将资源返回系统，此过程就是 XR Runtime 的生命周期。

```
GSXR_DEFINE_HANDLE(GSXR_Runtime)
```

6.2.1. 创建及初始 Runtime 实例

GSXR 提供 GSXR_CreateRuntime 函数进行 Runtime 实例的创建及初始，XR 应用程序必须输入自身的内容类型及操作系统的特定化结构成员，XR Runtime 必须根据应用程序的输入信息进行整个 XR 系统的创建及初始。

[枚举] GSXR_ApplicationType (应用类型)

名称	数值	描述
GSXR_ApplicationType_VR	1	表示 VR 应用
GSXR_ApplicationType_AR	2	表示 AR 应用

[结构体] GSXR_RuntimeCreateInfo (Runtime 初始信息)

```
typedef struct GSXR_RuntimeCreateInfo {
    void*                       next;
    GSXR_ApplicationType        appType;
    const void*                 systemBinding;
} GSXR_RuntimeCreateInfo;
```

说明：

Runtime 初始信息结构体，用以创建并初始化 Runtime 实例，next 作为预留指针会为加载不同厂商头手传参，load 会以 json 格式传入头手动态库的路径，Runtime 内部动态加载头手设备库中的方法达成使用不同头手进行组合使用。

Json 格式：

"hmdPath":["/xxx/xxx.so"], //hmd 动态库路径

"handPath":["/xxxx/xxx.so"] //hand 动态库路径

成员：

next 预留指针 支持动态加载机制传入 json 格式字符串头手动态库位置

appType 指定 XR 应用类型

systemBinding 操作系统的特定结构 (在移动设备系统, systemBinding 指向 GSXR_AndrosSystemBinding)

[结构体] GSXR_AndrosSystemBinding (移动设备系统特定信息)

```
typedef struct GSXR_AndrosSystemBinding {
    void*                       next;
    void*                       androsAppVM;
    void*                       androsAppActivity;
} GSXR_AndrosSystemBinding;
```

说明:

移动设备系统特定信息结构体, 用以提供操作系统特定化句柄给 Runtime

成员:

next 预留指针

androsAppVM 指向 JNI JavaVM 结构体

androsAppActivity Activity 的 jobject 参考

androsAppSurface 应用的 surface

[函数] GSXR_CreateRuntime (创建 Runtime)

```
GSXR_Result GSXR_CreateRuntime (
    const GSXR_RuntimeCreateInfo*       createInfo,
    GSXR_Runtime*                       runtime);
```

说明:

创建并初始 GSXR Runtime 实例, 始能 XR Runtime 运行平台

参数:

[in] createInfo 指向 GSXR_RuntimeCreateInfo 的结构体, 内含 Runtime 初始信息

[out] 指向 GSXR_Runtime 句柄的指针, runtime 实例创建成功输出一个有效句柄, 创建失败输出 GSXR_NULL_HANDLE

返回值:

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_ApplicationType_Unsupported

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

备注：

无

[基础能力规范] GSXR_CreateRuntime

[参数检验]

- `createInfo` 必须是指向 GSXR_RuntimeCreateInfo 的有效指针。
- `runtime` 必须是指向 GSXR_Runtime 句柄的有效指针。

[函数实现]

- Runtime 根据 createInfo 输入的初始信息完成 Runtime 初始化，并输出有效的 runtime 句柄。若创建失败则输出 GSXR_NULL_HANDLE。

[返回值]

成功

- `GSXR_Result_Success` Runtime 创建成功。

失败

- `GSXR_Error_Operation_Failed` Runtime 创建失败。
- `GSXR_Error_ApplicationType_Unsupported` 设备未支持该应用类型。
- `GSXR_Error_Argument_Invalid` createInfo 为 nullptr 或检验为无效的输入参数。或 runtime 为 nullptr。
- `GSXR_Error_Out_Of_Memory` 内存溢出。

6.2.2. 终止及销毁 Runtime 实例

GSXR 提供 GSXR_DestroyRuntime 函数进行 Runtime 实例的终止及销毁。

[函数] GSXR_DestroyRuntime (销毁 Runtime)

```
GSXR_Result GSXR_DestroyRuntime (
    GSXR_Runtime                          runtime);
```

说明：

终止并销毁 GSXR Runtime 实例

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_DestroyRuntime

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。

[函数实现]

- Runtime 根据输入的 runtime 句柄终止并销毁 Runtime。

[返回值]

成功

- `GSXR_Result_Success` Runtime 销毁成功。

失败

- `GSXR_Error_Operation_Failed` Runtime 销毁失败。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Call_Flow_Invalid` 调用此函数前未先行调用 GSXR_CreateRuntime。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

6.3. XR Runtime 属性

在成功创建并初始 XR Runtime 之后，XR 应用程序可经由 Runtime 句柄调用 GSXR_GetRuntimeProperties 函数获取 Runtime 属性信息，包括 Runtime 实现的 GSXR 版本号、Runtime 自身版本号、以及 Runtime 实现者名称。

[结构体] GSXR_RuntimeProperties (Runtime 属性信息)

```
typedef struct GSXR_RuntimeProperties {
    void*                       next;
    GSXR_Version                gsxrVersion;
    GSXR_Version                runtimeVersion;
    char                        runtimeName[GSXR_COMMON_STRING_MAX_SIZE];
} GSXR_RuntimeProperties;
```

说明：

Runtime 属性信息结构体，应用程序可用以获取 Runtime 详细属性

成员：

next 预留指针

gsxrVersion Runtime 实现的 GSXR 规范版本号

runtimeVersion Runtime 自身版本号

runtimeName Runtime 实现者名称

[函数] GSXR_GetRuntimeProperties (获取 Runtime 属性)

```
GSXR_Result GSXR_GetRuntimeProperties (
    GSXR_Runtime                        runtime,
    GSXR_RuntimeProperties*             runtimeProperties);
```

说明：

获取 GSXR Runtime 实现者的属性信息

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] runtimeProperties 指向 GSXR_RuntimeProperties 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_API_Unsupported

GSXR_Error_Handle_Invalid

备注：

无

[基础能力规范] GSXR_GetRuntimeProperties

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `runtimeProperties` 必须是指向 GSXR_RuntimeProperties 结构的有效指针。

[函数实现]

- Runtime 将自身属性信息填入 runtimeProperties 中输出。

[返回值]

成功

- `GSXR_Result_Success` Runtime 获取成功。

失败

- `GSXR_Error_Operation_Failed` Runtime 获取失败。
- `GSXR_Error_API_Unsupported` Runtime 未实现此 API。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。

6.4. 事件轮询

GSXR_Event 为 GSXR 事件结构体，从事件轮询函数 GSXR_PollEvent 获取事件指针后，需要根据该事件类型 GSXR_EventType，进而获取存于 GSXR_EventData 中的事件成员数据。

[结构体] GSXR_Event（事件信息）

```
typedef struct GSXR_Event {
   void*               next;
   GSXR_EventType      type;
   GSXR_Time           timestamp;
   uint8_t             buffer[GSXR_EVENT_BUFFER_MAX_SIZE];
   GSXR_EventData      gsxrEventData;
} GSXR_Event;
```

说明：

Runtime 事件信息结构体，应用程序可通过事件轮询函数获取 Runtime 即时事件

成员：

next 预留指针

type 事件类型

timestamp 事件发生时间戳

buffer 预留足够长度的事件数据缓存，以供储存事件指针转型后的事件结构成员

事件结构指针需要根据事件类型 GSXR_EventType 转为对应的事件结构指针，以下说明各事件类型的分类及其对应事件结构体。

[Union] GSXR_EventData（事件数据信息）

```
typedef union GSXR_EventData {
   GSXR_DeviceStateEventData      gsxr_devicestate_eventdata;
   GSXR_DeviceInputEventData      gsxr_deviceinput_eventdata;
   GSXR_RendererStateEventData    gsxr_rendererstate_eventdata;
   GSXR_XrFeatureStateEventData   gsxr_xrfeaturestate_eventdata;
} GSXR_EventData;
```

1. Common Events

XR Runtime 通用的事件属于此类，通常用来告知 XR 应用程序某种 XR Runtime 状态，事件名称以 GSXR_EventType_Common_为前缀。

[枚举] GSXR_EventType（通用事件）

名称	数值	描述
GSXR_EventType_Common_SpaceOrigin_Located	1000	应用空间原点设置完成
GSXR_EventType_Common_SpaceOrigin_Relocalized	1001	系统重置空间原点
GSXR_EventType_Common_SpaceBounds_Changed	1002	系统更动空间边界

2. Device State Events

XR 设备状态事件属于此类,通常用来告知 XR 应用程序的某种 XR 设备状态，事件名称以 GSXR_EventType_DeviceState_为前缀。GSXR_Event 根据该事件类型获取 GSXR_EventData 下的 GSXR_DeviceStateEventData，以获取该事件的设备类型 GSXR_DeviceType。

[结构体] GSXR_DeviceStateEvent (设备状态事件信息)

```
typedef struct GSXR_DeviceStateEventData {
   GSXR_DeviceType     deviceType;  // 设备类型
} GSXR_DeviceStateEventData;
```

[枚举] GSXR_EventType (设备状态事件)

名称	数值	描述
GSXR_EventType_DeviceState_Connected	2000	设备连接
GSXR_EventType_DeviceState_Disconnected	2001	设备断开
GSXR_EventType_DeviceState_InputMapping_Paired	2002	输入部件配对完成
GSXR_EventType_DeviceState_TrackingMode_Changed	2003	设备跟踪模式变换

3. Device Input Events

XR 设备输入事件属于此类，通常用来告知 XR 应用程序某种 XR 设备的输入事件，事件名称以 GSXR_EventType_DeviceInput_为前缀。GSXR_Event 根据该事

件类型获取 GSXR_EventData 下的 GSXR_DeviceInputEventData，以获取事件的设备类型 GSXR_DeviceType、GSXR_Device_InputType 设备输入类型及其输入部件标识号 InputId。

[结构体] GSXR_DeviceInputEventData（设备输入事件信息）

```
typedef struct GSXR_DeviceInputEventData {
   GSXR_DeviceType          deviceType;
   GSXR_Device_InputType    inputType;
   int                      inputId;         // 输入部件标识号
} GSXR_DeviceInputEventData;
```

[枚举] GSXR_EventType（设备输入事件）

名称	数值	描述
GSXR_EventType_DeviceInput_Clicked	3000	起始点击
GSXR_EventType_DeviceInput_Unclicked	3001	结束点击
GSXR_EventType_DeviceInput_Touched	3002	起始触摸
GSXR_EventType_DeviceInput_Untouched	3003	结束触摸
GSXR_EventType_DeviceInput_LeftToRight_Swiped	3004	左至右滑动
GSXR_EventType_DeviceInput_RightToLeft_Swiped	3005	右至左滑动
GSXR_EventType_DeviceInput_TopToBottom_Swiped	3006	上至下滑动
GSXR_EventType_DeviceInput_BottomToTop_Swiped	3007	下至上滑动

4. Renderer State Events

渲染器状态事件属于此类，通常用来告知 XR 应用程序某种渲染器状态，事件名称以 GSXR_EventType_RendererState_为前缀。GSXR_Event 根据该事件类型获取 GSXR_EventData 下的 GSXR_RendererStateEventData，以获取事件的渲染

器句柄 GSXR_Renderer，视图类型 GSXR_ViewType，纹理队列句柄 GSXR_TextureQueue，以及渲染循环句柄 GSXR_RenderLooper。

[结构体] GSXR_RendererStateEventData（渲染器状态事件信息）

```
typedef struct GSXR_RendererStateEventData {
    GSXR_Renderer           renderer;       // 渲染器句柄
    GSXR_ViewType           viewType;       // 视图类型
    GSXR_TextureQueue       textureQueue;   // 纹理队列句柄
    GSXR_RenderLooper       looper;         // 渲染循环句柄
} GSXR_RendererStateEventData;
```

[枚举] GSXR_EventType（渲染器状态事件）

名称	数值	描述
GSXR_EventType_RendererState_Renderer_Created	4000	渲染器创建
GSXR_EventType_RendererState_Renderer_Destroyed	4001	渲染器销毁
GSXR_EventType_RendererState_TextureQueue_Created	4002	纹理队列创建
GSXR_EventType_RendererState_TextureQueue_Destroyed	4003	纹理队列销毁
GSXR_EventType_RendererState_Looper_Available	4004	渲染循环状态可用
GSXR_EventType_RendererState_Looper_Unavailable	4005	渲染循环状态不可用
GSXR_EventType_RendererState_Looper_Starting	4006	渲染循环起始中
GSXR_EventType_RendererState_Looper_Stopping	4007	渲染循环结束中
GSXR_EventType_RendererState_Looper_Ready	4008	渲染循环已备妥
GSXR_EventType_RendererState_Looper_Unready	4009	渲染循环未备妥
GSXR_EventType_RendererState_Looper_Focused	4010	渲染循环获取输入焦点
GSXR_EventType_RendererState_Looper_Unfocused	4011	渲染循环失去输入焦点
GSXR_EventType_RendererState_ViewSet_Changed	4012	视图集内容变更

GSXR_EventType_RendererState_ViewSet_Added	4013	视图集添加
GSXR_EventType_RendererState_ViewSet_Removed	4014	视图集删除

5. Feature State Events

功能状态事件属此类，事件名称以 GSXR_EventType_FeatureState_为前缀。GSXR_Event 根据该事件类型获取 GSXR_EventData 下的 GSXR_RendererStateEventData，以获取事件的特性类型 GSXR_XrFeatureType。

[结构体] GSXR_XrFeatureStateEvent (特性状态事件信息)

```
typedef struct GSXR_XrFeatureStateEvent {
   GSXR_XrFeatureType     featureType;  // 特性类型
} GSXR_XrFeatureStateEvent;
```

[枚举] GSXR_EventType (特性状态事件)

名称	数值	描述
GSXR_EventType_FeatureState_Unavailable	5000	特性状态不可用
GSXR_EventType_FeatureState_Available	5001	特性状态可用
GSXR_EventType_FeatureState_Starting	5002	特性状态起始中
GSXR_EventType_FeatureState_Working	5003	特性状态运作中
GSXR_EventType_FeatureState_Stopping	5004	特性状态结束中

了解上述定义的事件结构及事件类型后，XR 应用程序便可通过事件轮询函数获取事件队列中的事件。事件队列是一个固定大小的储存体，Runtime 触发的事件便记录于队列中，队列大小由 Runtime 自行定义，但是不能在 XR 应用程序在以合理频率取用的情况下造成队列溢出。

[注意]

Runtime 的实现必须考虑 XR 应用程序在渲染循环中定期轮询队列时，不能出现队列溢出。队列溢出只会出现在 XR 应用程序未能如期从队列中取出事件的状况，且此状况可能造成 XR 应用程序非预期的运行结果。

[函数] GSXR_PollEvent（轮询事件队列获取事件）

```
GSXR_Result GSXR_PollEvent(
    GSXR_Runtime                    runtime,
    GSXR_Event*                     event);
```

说明：

从事件队列中获取 GSXR 事件，返回 GSXR_Result_Success 代表队列中尚有事件，需再次调用函数获取下一事件，直至返回 GSXR_Result_Event_Unavailable 代表队列中已无新事件

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] event 指向 GSXR_Event 结构体

返回值：

GSXR_Result_Success

GSXR_Result_Event_Unavailable

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_PollEvent

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- **`event` 必须是指向 GSXR_Event 结构的有效指针。**

[函数实现]

- Runtime 从事件队列中取出最早存入的事件，填入 event 中输出。
- Runtime 的实现，在应用程序定期正确轮询队列的情形下，不会出现队列溢出。

[返回值]

成功

- `GSXR_Result_Success` 事件获取成功，队列中尚有事件。
- `GSXR_Result_Event_Unavailable` 事件队列中已无事件。

失败

- `GSXR_Error_Operation_Failed` 事件获取失败。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

7. XR 设备函数说明

XR 内容应用必须搭配对应的 XR 设备硬件，才能将应用内容及用户交互行为反映在设备上进行交流体验。因此，XR Runtime 运行时，必须确保 XR 设备正常运作，才能从设备端提供 XR 应用程序所需的输入事件、空间位姿、设备信息、以及振动反馈。

本章描述 GSXR 的设备相关函数，包含设备类型及输入部件的定义及判读方式，从事件轮询函数中获取设备连线及输入事件，如何经过函数操控设备输入输出功能，空间位姿的获取和重置，以及获取设备相关信息等。

[XR 设备接口设计]

在同一 XR Runtime 实例中，一种设备类型只允许接入一个设备，并不允许接入多个设备（例如：Runtime 允许一个 VR 头显配上左右手柄各一个；但不允许一个 VR 头显配上两个右手手柄。），且设备连线是一种被动式的操作行为，当设备连线时 XR Runtime 便必须为此设备完成初始化，不需要经由应用程序主动调用函数为其初始。

故本章设备函数全部以设备类型为输入参数识别操作对象，不若 XR Runtime 及 XR 渲染器是以特定句柄为函数输入参数进行对象识别。主要函数设计差异处于此处先行说明。

7.1. 设备类型

XR 设备根据主要交互行为区分设备类型，且设备根据自身的设计及特性，也可以在设备上接入数目及类型不等的输入部件。因此，XR Runtime 必须将当前连线设备的类型及其输入部件配置传递给 XR 应用进行相应的处理，增加应用对于设备差异的兼容性。本节只描述设备类型，输入部件将于 7.4 节中介绍。

[枚举] GSXR_DeviceType（设备类型）

名称	数值	描述
GSXR_DeviceType_HMD_AR	1	AR 头戴式显示器
GSXR_DeviceType_HMD_VR	2	VR 头戴式显示器
GSXR_DeviceType_Controller_Right	3	右手手柄控制器
GSXR_DeviceType_Controller_Left	4	左手手柄控制器
GSXR_DeviceType_BaseStation	5	定位基站模组
GSXR_DeviceType_LeftFoot	6	左脚跟踪模组
GSXR_DeviceType_RightFoot	7	右脚跟踪模组
GSXR_DeviceType_LeftShoulder	8	左肩跟踪模组
GSXR_DeviceType_RightShoulder	9	右肩跟踪模组
GSXR_DeviceType_Waist	10	腰部跟踪模组
GSXR_DeviceType_Chest	11	胸部跟踪模组
GSXR_DeviceType_Threadmil	12	threadmil 跟踪模组
GSXR_DeviceType_Gamepad	13	gamepad 控制器

GSXR_DeviceType_Tracker	64	独立跟踪模组

[函数] GSXR_GetSupportedDevices (获取支持的设备类型)

```
GSXR_Result GSXR_GetSupportedDevices (
    GSXR_Runtime          runtime,
    GSXR_Flags64*         deviceTypeFlags);
```

说明：

获取当前支持的所有设备类型

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] deviceTypeFlags 指向 GSXR_Flags64 数值的指针，数值为 GSXR_DeviceType 的位掩码，输出可支持的所有设备类型

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSupportedDevices

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceTypeFlags` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 需要根据本身能力枚举当前运行平台所支持的所有设备，将所支持设备类型对应的 GSXR_DeviceType 数值进行 OR 运算后的位掩码填入 deviceTypeFlags 中输出。

[返回值]

成功

- `GSXR_Result_Success` 设备类型获取成功。

失败

- GSXR_Error_Operation_Failed 设备类型获取失败。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

7.2. 设备连线状态

在 XR 应用程序开始使用 XR 设备之前，XR 设备必须先进行连线并完成初始化，此段流程由 XR 设备插件与 XR Runtime 交互进行，并在 XR Runtime 完成设备初始化后发送 GSXR_EventType_DeviceState_Connected 事件通知 XR 应用程序，随后 XR 应用程序始能对于该连线设备进行其设备函数操作。

反之，当 XR 设备断开连线，XR Runtime 也需要释放该设备的资源，而后发送 GSXR_EventType_DeviceState_Disconnected 事件通知 XR 应用程序。

[基础能力规范] 设备连线事件

[功能实现]

- GSXR_EventType_DeviceState_Connected 设备连接，XR Runtime 完成设备初始化后发送 GSXR_EventType_DeviceState_Connected 事件进事件队列。
- GSXR_EventType_DeviceState_Disconnected 设备断开，XR Runtime 完成资源释放后发送 GSXR_EventType_DeviceState_Disconnected 事件进事件队列。

除被动接收设备连线事件之外，GSXR 也可以提供 GSXR_IsDeviceConnected 函数供 XR 应用程序主动获取当前的设备连线状态。

[函数] GSXR_IsDeviceConnected (获取设备连线状态)

```
GSXR_Result GSXR_IsDeviceConnected (
    GSXR_Runtime          runtime,
```

```
    GSXR_DeviceType        deviceType,
    GSXR_Bool32*           connected);
```

说明：

获取设备连线状态

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[out] connected 指向 GSXR_Bool32 数值的指针，数值输出 GSXR_TRUE 表示已连线，GSXR_FALSE 表示未连线

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_IsDeviceConnected

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `connected` 必须是指向 GSXR_Bool32 数值的有效指针。

[函数实现]

- Runtime 须根据当前 deviceType 的连线状态输出 connected（GSXR_TRUE 表示已连线，GSXR_FALSE 表示未连线）。

[返回值]

成功

- `GSXR_Result_Success` 设备连线状态获取成功。

失败

- `GSXR_Error_Operation_Failed` 设备连线状态获取失败。

- GSXR_Error_Function_Unsupported deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceType 未定义于 GSXR_DeviceType 中。或 connected 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

7.3. 显示设备标识号

具备显示能力的 XR 设备（例如：VR 头戴式显示器），XR 应用程序必须从中获取显示标识号 GSXR_DisplayId,供 XR Runtime 创建渲染器使用,XR Runtime 将根据此标识号将 XR 渲染帧输出到此显示设备。

[函数] GSXR_GetDisplay (获取显示设备标识号)

```
GSXR_Result GSXR_GetDisplay (
      GSXR_Runtime            runtime,
      GSXR_DeviceType         deviceType,
      GSXR_DisplayId*         displayId);
```

说明：

获取显示设备标识号, 只能从具备显示屏能力的设备类型获取

参数：

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[out] displayId 指向 GSXR_DisplayId 标识号的指针, 若设备具显示屏能力则输出显示标识号, 否则输出 GSXR_NULL_DISPLAY_ID

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetDisplay

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `displayId` 必须是指向 GSXR_DisplayId 数值的有效指针。

[函数实现]

- Runtime 判断 deviceType 具备显示屏能力并输出其对应 displayId。
- 此 displayId 将用于 Runtime 渲染器视图识别输出显示设备使用。

[返回值]

成功

- `GSXR_Result_Success` 显示标识号获取成功。

失败

- `GSXR_Error_Operation_Failed` 显示标识号获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备，或该设备不具备显示标识号。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 displayId 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

7.4. 输入部件

输入部件是 XR 设备在硬件设计上提供用户输入的交互人机界面，XR 应用程序根据自身的设计需求使用相应的输入部件，提供用户与内容进行交互。同样地，

XR 设备所具备的输入部件配置将会影响应用内容的操作行为，此节将详细介绍 GSXR 关于输入部件的相关内容。

7.4.1. 输入部件的标识号与类型

通常头显设备与手柄设备所提供的输入部件并不相同，即便为同类型设备，各设备对于输入部件的硬件配置也不一致。为使 XR 应用程序明确获知 XR 设备的输入部件能力及配置，GSXR 提供以下输入部件定义及函数，XR 应用程序可根据设备不同的输入部件能力，设计相对应的内容交互方式，提高 XR 应用在不同输入设备上的泛用性。

[枚举] GSXR_Touch_InputId（输入部件标识号，如 HTC 类型手柄）

名称	数值	描述
GSXR_Touch_InputId_System	1	系统键
GSXR_Touch_InputId_Menu	2	选单键
GSXR_Touch_InputId_Grip	3	抓握键
GSXR_Touch_InputId_Trigger	4	扳机
GSXR_Touch_InputId_TrackPad	5	触摸板
GSXR_Touch_InputId_DPad_Left	6	十字键-左
GSXR_Touch_InputId_DPad_Up	7	十字键-上
GSXR_Touch_InputId_DPad_Right	8	十字键-右
GSXR_Touch_InputId_DPad_Down	9	十字键-下

[枚举] GSXR_Joystick_InputId (输入部件标识号，如 OC Quest 类型手柄)

名称	数值	描述
GSXR_Joystick_InputId_A	1	A
GSXR_Joystick_InputId_B	2	B
GSXR_Joystick_InputId_X	3	X
GSXR_Joystick_InputId_Y	4	Y
GSXR_Joystick_InputId_System	5	系统键
GSXR_Joystick_InputId_Menu	6	选单键
GSXR_Joystick_InputId_Grip	7	抓握键
GSXR_Joystick_InputId_Trigger	8	扳机
GSXR_Joystick_InputId_Thumstick	9	摇杆
GSXR_Joystick_InputId_Dpad_Left	10	十字键-左
GSXR_Joystick_InputId_Dpad_Up	11	十字键-上
GSXR_Joystick_InputId_Dpad_Right	12	十字键-右
GSXR_Joystick_InputId_Dpad_Down	13	十字键-下

[枚举] GSXR_Gamepad_InputId (输入部件标识号，如 Gamepad 类型手柄)

名称	数值	描述
GSXR_Gamepad_InputId_A	1	A
GSXR_Gamepad_InputId_B	2	B
GSXR_Gamepad_InputId_X	3	X
GSXR_Gamepad_InputId_Y	4	Y
GSXR_Gamepad_InputId_Left_Thumbstick	5	摇杆

GSXR_Gamepad_InputId_Right_Thumbstick	6	摇杆
GSXR_Gamepad_InputId_Left_Thumbstick_Up	7	十字键-上
GSXR_Gamepad_InputId_Left_Thumbstick_Down	8	十字键-下
GSXR_Gamepad_InputId_Left_Thumbstick_Left	9	十字键-左
GSXR_Gamepad_InputId_Left_Thumbstick_Right	10	十字键-右
GSXR_Gamepad_InputId_Right_Thumbstick_Up	11	十字键-上
GSXR_Gamepad_InputId_Right_Thumbstick_Down	12	十字键-下
GSXR_Gamepad_InputId_Right_Thumbstick_Left	13	十字键-左
GSXR_Gamepad_InputId_Right_Thumbstick_Right	14	十字键-右
GSXR_Gamepad_InputId_Dpad_Up	15	十字键-上
GSXR_Gamepad_InputId_Dpad_Down	16	十字键-下
GSXR_Gamepad_InputId_Dpad_Left	17	十字键-左
GSXR_Gamepad_InputId_Dpad_Right	18	十字键-右
GSXR_Gamepad_InputId_Left_Trigger	19	扳机
GSXR_Gamepad_InputId_Left_Shoulder	20	Shoulder
GSXR_Gamepad_InputId_Right_Trigger	21	扳机
GSXR_Gamepad_InputId_Right_Shoulder	22	Shoulder

[枚举] GSXR_Hmd_InputId (输入部件标识号)

名称	数值	描述
GSXR_Hmd_InputId_Enter	1	确认按键
GSXR_Hmd_InputId_Back	2	返回按键
GSXR_Hmd_InputId_Volume_Up	3	音量+
GSXR_Hmd_InputId_Volume_Down	4	音量-
GSXR_Hmd_InputId_Home	5	Home 按键

[枚举] GSXR_InputId_InputType (输入类型)

名称	数值	描述
GSXR_InputId_InputType_Click	0x00000001	点击类
GSXR_InputId_InputType_Touch	0x00000002	触摸类
GSXR_InputId_InputType_Analog_1D	0x00000004	类比类 - 一维数据
GSXR_InputId_InputType_Analog_2D	0x00000008	类比类 - 二维数据

[枚举] GSXR_Device_InputType (设备输入类型)

名称	数值	描述
GSXR_NO_Supported_InputType	1	设备没有支持的输入类型
GSXR_Touch	2	触摸版类型控制器输入类型，如 HTC 控制器
GSXR_Joystick	3	摇杆类型控制器输入类型，如 Oculus Quest 控制器
GSXR_Gamepad	4	Gamepad 控制器输入类型
GSXR_Hmd	5	Hmd 头部输入类型

[函数] GSXR_GetSupportedDeviceInputType (获取设备支持的输入类型)

```
GSXR_Result GSXR_GetSupportedDeviceInputType (
```

```
    GSXR_Runtime            runtime,
    GSXR_DeviceType         deviceType,
    GSXR_Flags64*           deviceInputTypeFlag);
```

说明：

获取当前设备支持的输入类型

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[out] deviceInputTypeFlag 指向 GSXR_Flags64 数值的指针，数值为 GSXR_Device_InputType ，输出设备支持的输入类型，

如果设备类型是 GSXR_DeviceType_Controller_Left，则 deviceInputType 为 GSXR_Touch 或者 GSXR_Joystick 输入类型之一，

如果设备类型是 GSXR_DeviceType_Controller_Right,则 deviceInputType 为 GSXR_Touch 或者 GSXR_Joystick 输入类型之一，

如果设备类型是 GSXR_DeviceType_Gamepad,则 deviceInputType 是 GSXR_Gamepad 输入类型，

如果设备类型是 GSXR_DeviceType_Hmd_VR,则 deviceInputType 是 GSXR_Hmd 输入类型，

对于其他设备类型暂未开放输入类型，则 deviceInputType 为 GSXR_NO_Supported_InputType

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Function_Unsupported

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSupportedDeviceInputType

[参数检验]

- runtime 必须是有效的 GSXR_Runtime 句柄。
- deviceType 必须是有效的 GSXR_DeviceType
- deviceTypeFlags 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 需要根据本身特点,将设备支持的输入类型 GSXR_Device_InputType 数值填入 deviceInputTypeFlag 中输出。

[返回值]

成功

- GSXR_Result_Success 设备类型获取成功。

失败

- GSXR_Error_Operation_Failed 设备类型获取失败。
- GSXR_Error_Function_Unsupported deviceType 没有支持的输入类型
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceTypeFlags 为 null
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

[函数] GSXR_GetInputIdConfiguration (获取输入部件具备的输入类型配置)

```
GSXR_Result GSXR_GetInputIdConfiguration (
      GSXR_Runtime           runtime,
      GSXR_DeviceType        deviceType,
      GSXR_Device_InputType  inputType
      int                    inputId,
      GSXR_Flags64*          configuration);
```

说明:

获取设备输入部件所具备的输入类型配置

参数:

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

目前只开放了 GSXR_DeviceType_Controller_Left、GSXR_DeviceType_Controller_Right、GSXR_DeviceType_Gamepad 等设备具备输入类型

[in] inputType 指定设备输入类型

如果设备类型是 GSXR_DeviceType_Controller_Left，则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Controller_Right,则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Gamepad,则 inputType 为 GSXR_Gamepad 类型

此参数必须与 GSXR_GetSupportedDeviceInputType 保持一致

[in] inputId 指定输入部件

如果设备类型是 GSXR_DeviceType_Controller_Left，则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型下的 inputId

如果设备类型是 GSXR_DeviceType_Controller_Right,则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型下的 inputId

如果设备类型是 GSXR_DeviceType_Gamepad,则 inputType 为 GSXR_Gamepad 类型下的 inputId

[out] configuration 指向 GSXR_Flags64 数值的指针，数值为 GSXR_InputId_InputType 的位掩码，输出此输入部件所具备的输入类型配置

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetInputIdConfiguration

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。

- deviceType 必须是有效的 GSXR_DeviceType 类型。
- inputType 必须是有效的 GSXR_Device_InputType 类型，并且必须与 GSXR_GetSupportedDeviceInputType 保持一致
- inputId 必须是有效的 GSXR_InputId 类型。
- configuration 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 根据设备部件配置，获取 deviceType 设备的 inputId 部件具备何种输入类型能力，并将对应类型的 GSXR_InputType 进行 OR 运算后的位掩码填入 configuration 中输出。

[返回值]

成功

- GSXR_Result_Success 输入部件配置获取成功。

失败

- GSXR_Error_Operation_Failed 输入部件配置获取失败。
- GSXR_Error_Function_Unsupported deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。或 inputId 未定义于 GSXR_Device_InputType 输入类型下的 InputId，或者 InputType 未与 GSXR_GetSupportedDeviceInputType 保持一致。
- GSXR_Error_Device_Disconnected deviceType 尚未连线。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceType 未定义于 GSXR_DeviceType 中。或 inputId 未定义于 GSXR_Device_InputType 输入类型下的 InputId 中。或 configuration 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

7.4.2. 获取输入状态数据

在 XR Runtime 完成输入部件的配对之后，Runtime 便可允许 XR 设备的输入部件数据传递至 XR 应用程序中。XR 应用程序获取 XR 设备输入部件状态数据的方式有二：

1. 被动式获取单一事件

XR 应用程序调用 GSXR_PollEvent 函数获取事件，当事件类型为以下类型，表示 XR 设备的输入部件触发了输入事件，XR 应用程序必需将函数输出的 GSXR_Event 指针转型为 GSXR_DeviceInputEvent，从中获取触发输入事件的设备类型及输入部件标识号。

点击事件

- GSXR_EventType_DeviceInput_Clicked
- GSXR_EventType_DeviceInput_Unclicked

触控事件

- GSXR_EventType_DeviceInput_Touched
- GSXR_EventType_DeviceInput_Untouched

滑动事件

- GSXR_EventType_DeviceInput_LeftToRight_Swiped
- GSXR_EventType_DeviceInput_RightToLeft_Swiped
- GSXR_EventType_DeviceInput_TopToBottom_Swiped
- GSXR_EventType_DeviceInput_BottomToTop_Swiped

[基础能力规范] 设备输入事件

[功能实现]

点击事件

- 设备输入部件支持点击功能
- `GSXR_EventType_DeviceInput_Clicked` 输入部件状态从“未点击”进入“点击”，Runtime 发送 GSXR_EventType_DeviceInput_Clicked 事件进事件队列。
- `GSXR_EventType_DeviceInput_Unclicked` 输入部件状态从“点击”进入“未点击”，Runtime 发送 GSXR_EventType_DeviceInput_Unclicked 事件进事件队列。

触摸事件

- 设备输入部件支持触摸功能
- GSXR_EventType_DeviceInput_Touched 输入部件状态从“未触摸”进入“触摸”，Runtime 发送 GSXR_EventType_DeviceInput_Touched 事件进事件队列。
- GSXR_EventType_DeviceInput_Untouched 输入部件状态从“触摸”进入“未触摸”，Runtime 发送 GSXR_EventType_DeviceInput_Untouched 事件进事件队列。

滑动事件

- 设备输入部件支持滑动功能
- GSXR_EventType_DeviceInput_LeftToRight_Swiped 输入轨迹为一次有效的左至右滑动，Runtime 发送 GSXR_EventType_DeviceInput_LeftToRight_Swiped 事件进事件队列。
- GSXR_EventType_DeviceInput_RightToLeft_Swiped 输入轨迹为一次有效的右至左滑动，Runtime 发送 GSXR_EventType_DeviceInput_RightToLeft_Swiped 事件进事件队列。
- GSXR_EventType_DeviceInput_TopToBottom_Swiped 输入轨迹为一次有效的上至下滑动，Runtime 发送 GSXR_EventType_DeviceInput_TopToBottom_Swiped 事件进事件队列。
- GSXR_EventType_DeviceInput_BottomToTop_Swiped 输入轨迹为一次有效的下至上滑动，Runtime 发送 GSXR_EventType_DeviceInput_BottomToTop_Swiped 事件进事件队列。

2. 主动式获取当前状态数据

GSXR 提供三类函数获取点击状态、触摸状态、及类比数据。

- 调用 GSXR_GetInputClickStates 获取指定输入部件下的所有点击状态。
- 调用 GSXR_GetInputTouchStates 获取指定输入部件下的所有触摸状态。
- 调用 GSXR_GetInputAnalogState 获取指定输入部件的类比数据。

[函数] GSXR_GetInputClickState (获取设备输入部件点击状态)

```
GSXR_Result GSXR_GetInputClickStates (
      GSXR_Runtime             runtime,
      GSXR_DeviceType          deviceType,
      GSXR_Device_InputType    inputType,
      GSXR_Flags64*            clicks);
```

说明：

获取设备输入部件的点击状态

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[in] inputType 指定输入类型

如果设备类型是 GSXR_DeviceType_Controller_Left，则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Controller_Right,则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Gamepad,则 inputType 为 GSXR_Gamepad 类型

此参数必须与 GSXR_GetSupportedDeviceInputType 保持一致

[out] clicks 指向 GSXR_Flags64 数值之指针，数值按输入类型下的输入 ID 为位掩码储存 InputId 的点击状态信息

如果设备类型是 GSXR_DeviceType_Controller_Left，则 clicks 是按 InputType 类型下的 InputId 为位掩码来存储点击状态信息

如果设备类型是 GSXR_DeviceType_Controller_Right，则 clicks 是按 InputType 类型下的 InputId 为位掩码来存储点击状态信息

如果设备类型是 GSXR_DeviceType_Gamepad，则 clicks 是按 InputType 类型下的 InputId 为位掩码来存储点击状态信息

如果设备类型是其他的设备，暂时未开发 GSXR_Device_InputType 定义，暂不支持

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

备注：

XR 应用需优先调用 GSXR_SetInputRequest 设置 XR 应用的设备输入请求

[基础能力规范] GSXR_GetInputClickStates

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- inputType 指定输入类型，根据 GSXR_GetSupportedDeviceInputType 获取，并且要求保持一致。
- `clicks` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 获取 deviceType 设备输入类型下的所有 InputId 输入部件的点击状态信息，以 InputId 为位掩码存储点击状态信息

[返回值]

成功

- `GSXR_Result_Success` 点击状态获取成功。

失败

- `GSXR_Error_Operation_Failed` 点击状态获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。或 inputType 未与 GSXR_GetSupportedDeviceInputType 保持一致。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 inputType 未定义于 GSXR_Device_InputType 中。或 clicks 为 nullptr。
- `GSXR_Error_Call_Flow_Invalid` 未先完成 GSXR_SetInputRequest 的调用。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[函数] GSXR_GetInputTouchStates (获取设备输入部件触摸状态)

```
GSXR_Result GSXR_GetInputTouchStates (
      GSXR_Runtime            runtime,
      GSXR_DeviceType         deviceType,
      GSXR_Device_InputType   inputType
      GSXR_Flags64*           touchs);
```

说明：

获取设备输入部件的触摸状态

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[in] inputType 指定输入类型

如果设备类型是 GSXR_DeviceType_Controller_Left，则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Controller_Right,则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Gamepad,则 inputType 为 GSXR_Gamepad 类型

此参数必须与 GSXR_GetSupportedDeviceInputType 保持一致

[out] touchs 指向 GSXR_Flags64 数值的指针，数值按输入类型下的输入 ID 为位掩码储存 InputId 的触摸状态信息

如果设备类型是 GSXR_DeviceType_Controller_Left，则 buttons 是按 InputType 类型下的 InputId 为位掩码来存储触摸状态信息

如果设备类型是 GSXR_DeviceType_Controller_Right，则 buttons 是按 InputType 类型下的 InputId 为位掩码来存储触摸状态信息

如果设备类型是 GSXR_DeviceType_Gamepad，则 buttons 是按 InputType 类型下的 InputId 为位掩码来存储触摸状态信息

如果设备类型是其他的设备，暂时未开发 GSXR_Device_InputType 定义，暂不支持

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

备注：

XR 应用需优先调用 GSXR_SetInputRequest 设置 XR 应用的设备输入请求

[基础能力规范] GSXR_GetInputTouchState

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- inputType 指定输入类型，根据 GSXR_GetSupportedDeviceInputType 获取，并且要求保持一致。
- `touchs` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 获取 deviceType 设备输入类型下的所有 inputId 输入部件的触摸状态填入 touchs 中输出，以 InputId 为位掩码存储触摸状态信息

[返回值]

成功

- `GSXR_Result_Success` 触摸状态获取成功。

失败

- `GSXR_Error_Operation_Failed` 触摸状态获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。或 inputType 未与 GSXR_GetSupportedDeviceInputType 保持一致。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 inputType 未定义于 GSXR_Device_InputType 中。或 touchs 为 nullptr。
- `GSXR_Error_Call_Flow_Invalid` 未先完成 GSXR_SetInputRequest 的调用。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[函数] GSXR_GetInputAnalogState（获取设备输入部件类比数据）

```
GSXR_Result GSXR_GetInputAnalogState (
      GSXR_Runtime              runtime,
      GSXR_DeviceType           deviceType,
      GSXR_Device_InputType     inputType,
      int                       inputId,
```

```
    GSXR_Vector2f*            analog);
```

说明：

获取设备输入部件的类比数据

参数：

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[in] inputType 指定输入类型

如果设备类型是 GSXR_DeviceType_Controller_Left，则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Controller_Right,则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型

如果设备类型是 GSXR_DeviceType_Gamepad,则 inputType 为 GSXR_Gamepad 类型

此参数必须与 GSXR_GetSupportedDeviceInputType 保持一致

[in] inputId 指定输入部件

如果设备类型是 GSXR_DeviceType_Controller_Left，则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型下的 InputId,并转成 int 类型

如果设备类型是 GSXR_DeviceType_Controller_Right,则 inputType 为 GSXR_Touch 或者 GSXR_Joystick 类型下的 InputId 并转成 int 类型

如果设备类型是 GSXR_DeviceType_Gamepad,则 inputType 为 GSXR_Gamepad 类型下的 InputId 并转成 int 类型

[out] analog 指向二维矢量 GSXR_Vector2f 结构体的指针, 数据范围见备注说明

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

备注：

1. XR 应用需优先调用 GSXR_SetInputRequest 设置 XR 应用的设备输入请求
2. analog 数据范围

a、 若输入部件的输入类型为 GSXR_InputType_Analog_1D，analog 数据范围为 0<=x<=1，y=0

b、 若输入部件的输入类型为 GSXR_InputType_Analog_2D，analog 数据范围为 -1<=x<=1，-1<=y<=1

[基础能力规范] GSXR_GetInputAnalogState

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- inputType 指定输入类型，根据 GSXR_GetSupportedDeviceInputType 获取，并且要求保持一致。
- `inputId` 必须是有效的 InputId 类型。
- `analog` 必须是指向 GSXR_Vector2f 结构的有效指针。

[函数实现]

- Runtime 获取 deviceType 设备的 inputId 输入部件的类比数据填入 analog 中输出。
- analog 数据必须从原类比数据等比转换为函数备注 2 要求的归一化数据范围。

[返回值]

成功

- `GSXR_Result_Success` 类比数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 类比数据获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。或 inputType 未与 GSXR_GetSupportedDeviceInputType 保持一致。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 inputType 未定义于 GSXR_Device_InputType 中。或 inputId 未定义于 InputId 中。或 analog 为 nullptr。
- `GSXR_Error_Call_Flow_Invalid` 未先完成 GSXR_SetInputRequest 的调用。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

7.5. 输出振动反馈

振动反馈是一种利用触觉体验与用户进行交互的方法，通常用于手柄设备。要实现振动反馈，XR 设备在硬件设计上必须配置振动模组，使 XR 应用程序在内容的运用上，可以在适当时机能够驱动 XR 设备输出振动反馈，用来强化 XR 内容的交互体验。

[结构体] GSXR_HapticVibration (振动反馈)

```
typedef struct GSXR_HapticVibration {
    void*                   next;
    GSXR_Duration           duration;
    uint32_t                frequency;
    float                   intensity;
} GSXR_HapticVibration;
```

说明：

振动反馈结构体，用来描述振动反馈的需求

成员：

next 预留指针

duration 振动时长，单位为 nanoseconds

frequency 振动次数

intensity 振动强度，数据介于 0 - 1

[函数] GSXR_TriggerVibration (触发设备振动反馈)

```
GSXR_Result GSXR_TriggerVibration (
        GSXR_Runtime                        runtime,
        GSXR_DeviceType                     deviceType,
        GSXR_Bool32                         manual,
        const GSXR_HapticVibration*         haptics);
```

说明：

触发设备振动反馈，可自行设置时长,次数,强度，也可以交由系统触发一个适合当前

设备强度适中的短振动

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[in] manual 指定手动或自动调控，GSXR_TRUE 表示 XR 应用自行调控振动反馈，GSXR_FALSE 表示交由系统触发一个适合当前设备强度适中的短振动

[in] haptics 指向 GSXR_HapticVibration 结构体的指针，manual 为 GSXR_FALSE 时可输入 nullptr

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_TriggerVibration

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `manual` 为 GSXR_TRUE 时，`haptics` 必须是指向 GSXR_HapticVibration 结构的有效指针。

[函数实现]

- manual 为 GSXR_TRUE 时，Runtime 根据输入的 haptics 要求，触发 deviceType 设备进行对应的振动反馈。
- manual 为 GSXR_FALSE 时，Runtime 触发一个适合当前 deviceType 设备强度适中的短振动。振动时长及强度由 Runtime 依设备定义。

[返回值]

成功

- GSXR_Result_Success 触发振动成功。

失败

- GSXR_Error_Operation_Failed 触发振动失败。
- GSXR_Error_Function_Unsupported deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备，或该设备不支持振动功能。
- GSXR_Error_Device_Disconnected deviceType 尚未连线。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceType 未定义于 GSXR_DeviceType 中。或 manual 为 GSXR_TRUE 时，haptics 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

7.6. 空间位姿

XR 设备除了输入输出功能外，还需要具备空间自由度（Degree Of Freedom, 简称 DOF）跟踪能力，XR 应用程序才可从中获取空间位姿（Pose）运用于内容交互上。空间位姿是基于空间原点（Origin）的相对位姿，而根据 XR 应用的内容特性不同，内容空间原点类型也可能会有不同。此外，具备位移跟踪能力的系统，为预防虚拟空间与现实空间的场域差异造成用户在跟踪期间意外碰撞现实空间物体的状况，设置空间边界（Bounds）也可以为 XR 空间跟踪系统提供必要的安全措施。

本节将详述 GSXR 对于空间位姿系统的相关定义。

7.6.1. 空间自由度跟踪能力

一般 XR 设备的空间跟踪能力区分两类型：（1） 3DOF：只具备旋转跟踪能力，（2）6DOF：兼备旋转及位移跟踪能力。XR 应用必须根据 XR 设备的空间跟踪能力调整内容交互方式，使内容更好地适配各种不同场域的运用。

[枚举] GSXR_DegreeOfFreedom (空间自由度跟踪能力)

名称	数值	描述
GSXR_DegreeOfFreedom_None	0	无空间跟踪能力
GSXR_DegreeOfFreedom_3DOF	1	只具备旋转跟踪能力
GSXR_DegreeOfFreedom_6DOF	2	兼备旋转及位移跟踪能力

[函数] GSXR_GetDegreeOfFreedom (获取设备空间自由度跟踪能力)

```
GSXR_Result GSXR_GetDegreeOfFreedom (
      GSXR_Runtime                runtime,
      GSXR_DeviceType             deviceType,
      GSXR_DegreeOfFreedom*       dof);
```

说明：

获取设备的空间自由度跟踪能力

参数：

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[out] dof 指向 GSXR_DegreeOfFreedom 数值的指针, 输出空间自由度跟踪能力

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetDegreeOfFreedom

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `dof` 必须是指向 GSXR_DegreeOfFreedom 数值的有效指针。

[函数实现]

- Runtime 获取 deviceType 设备的空间自由度跟踪能力填入 dof 中输出。

[返回值]

成功

- `GSXR_Result_Success` 空间自由度获取成功。

失败

- `GSXR_Error_Operation_Failed` 空间自由度获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 dof 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

7.6.2. 空间原点类型

要使各个 XR 设备同时运行于同一空间，各设备必须使用同一世界坐标，才能于同一场域中精确定位出设备间彼此的空间位姿。因此，决定世界坐标的空间原点位置是 XR 应用程序在获取设备位姿之前的首要步骤。XR 应用程序应根据内容特性决定空间原点设置的类型，Runtime 将根据应用设置的空间原点类型适配不同跟踪能力的 XR 设备。

GSXR 提供两种空间原点类型供 XR 应用程序设置，如下表所示。

[枚举] GSXR_SpaceOrigin（空间原点类型）

名称	数值	描述
GSXR_SpaceOrigin_OnHead	1	空间原点始于头戴设备本身 （通常用于“没有地板平面”的内容）
GSXR_SpaceOrigin_OnFloor	2	空间原点考虑身高，校准至地板上 （通常用于“有地板平面”的内容，需搭配设备的空间校准程序）

由空间自由度及空间原点类型的定义，可归纳出以下规则：

- 若 XR 应用的内容特性为“无需地板平面”，内容需求视图起始位置落于场景中某固定点，则此类型的内容适合设置为 GSXR_SpaceOrigin_OnHead。
- 若 XR 应用的内容特性为“具有地板平面”，内容需求于场景中的视图高度真实呈现与实际地板位置相对的设备高度，则此类型的内容适合设置为 GSXR_SpaceOrigin_OnFloor。

[函数] GSXR_SetSpaceOrigin（设置 XR 应用的空间原点及偏移量）

```
GSXR_Result GSXR_SetSpaceOrigin (
      GSXR_Runtime            runtime,
      GSXR_SpaceOrigin        space,
      const GSXR_Posef*       offset = nullptr);
```

说明：

设置 XR 应用的空间原点及偏移量

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] space 指定空间原点类型

[in] offset 指定空间原点偏移量，若 XR 应用无需自设偏移量可输入 nullptr

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_SetSpaceOrigin

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `space` 必须是有效的 GSXR_SpaceOrigin 类型。

[函数实现]

- Runtime 以当前头显位姿加上偏移量 offset（若 offset 不为 nullptr)为跟踪原点，依据 space 类型及当前设备空间自由度跟踪能力决定跟踪位姿输出逻辑，详见下方空间位姿适配表。
- 设置完成后，Runtime 发送 GSXR_EventType_Common_SpaceOrigin_Located 事件进入事件队列。

[返回值]

成功

- `GSXR_Result_Success` 空间原点设置成功。

失败

- `GSXR_Error_Operation_Failed` 空间原点设置失败。
- `GSXR_Error_Function_Unsupported` space 定义于 GSXR_SpaceOrigin 中，但 Runtime 不支持该空间原点类型。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` space 未定义于 GSXR_SpaceOrigin 中。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[空间位姿适配表]

	GSXR_SpaceOrigin_OnHead 内容	GSXR_SpaceOrigin_OnFloor 内容
3DOF	Runtime 头显位置以内容起始点为原	Runtime 头显位置自带一个预设高度，

头显	点，且总是位于此点。	且位姿总是位于此高度。
6DOF 头显	Runtime 头显位置以空间校准原点为原点输出相对位姿。	Runtime 头显位置以实际地板相对高度为位姿高度。

7.6.3. 空间姿态

决定空间原点类型及位姿之后，XR 应用程序便可开始获取设备空间姿态运用于内容中。空间姿态内中包含两种数据，其一为位姿，用以描述设备的位置及方向，另一为速度，用以描述设备的线速度及角速度。并且跟踪系统算法根据现实环境的跟踪条件优劣，运算出的姿态数据可能失准，所以位姿及速度各具备一个布尔值描述当前数据是否为有效，若内容获取到无效数据便需要忽视该数据，以避免场景呈现不正常的运动行为。

[结构体] GSXR_PoseState（空间姿态）

```
typedef struct GSXR_PoseState {
    GSXR_Bool32             isPoseValid;
    GSXR_Posef              pose;
    GSXR_Bool32             isVelocityValid;
    GSXR_Velocityf          velocity;
} GSXR_PoseState;
```

说明：

空间姿态结构体，用以描述位姿及速度数据

成员：

isPoseValid GSXR_TRUE 表示位姿有效，GSXR_FALSE 表示位姿无效

pose 位姿结构体

isVelocityValid GSXR_TRUE 表示速度有效，GSXR_FALSE 表示速度无效

velocity 速度结构体

[函数] GSXR_GetPoseState（获取设备于指定时间的位姿及速度数据）

```
GSXR_Result GSXR_GetPoseState (
      GSXR_Runtime              runtime,
      GSXR_DeviceType           deviceType,
      GSXR_Time                 specifiedTime,
      GSXR_PoseState*           poseState);
```

说明：

获取设备于指定时间的位姿及速度数据

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[in] specifiedTime 指定时间

[out] poseState 指向 GSXR_PoseState 结构体的指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetPoseState

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `specifiedTime` 必须是有效的时间戳数值。
- `poseState` 必须是指向 GSXR_PoseState 结构的有效指针。

[函数实现]

- Runtime 获取 deviceType 设备的当前位姿及速度，并根据预测算法预测 specifiedTime

的姿态填入 poseState 输出。

[返回值]

成功

- `GSXR_Result_Success` 空间姿态获取成功。

失败

- `GSXR_Error_Operation_Failed` 空间姿态获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 specifiedTime 为一无效或算法无法支持的时间戳数值。或 poseState 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

7.6.4. 特殊跟踪系统事件

跟踪系统在某些情况下可能发生位姿偏移，在此情形下，XR 应用获取的空间位姿也将随之失准，影响内容体验。此时，跟踪系统根据产品行为定义可进行空间重置，重新校准位姿偏移量，改善内容体验。当跟踪系统进行空间（原点）重置时，为使 XR 应用程序获知空间原点已被系统重置，XR Runtime 必须发送 GSXR_EventType_Common_SpaceOrigin_Relocalized 事件通知应用，应用程序可根据本身内容特性进行相应处理。

[基础能力规范] 系统重置空间原点事件

[功能实现]

- `GSXR_EventType_Common_SpaceOrigin_Relocalized` 系统重置空间原点，XR Runtime 发送 GSXR_EventType_Common_SpaceOrigin_Relocalized 事件进入事件队列。

某些设备的跟踪系统，具有动态变换 3DOF / 6DOF 跟踪模式的能力，当跟踪模式变换时，XR 应用程序的内容操作行为通常也需要跟随新跟踪模式做相应调整。此时，为使 XR 应用程序获知跟踪模式已被系统变换，XR Runtime 必须发送 GSXR_EventType_DeviceState_TrackingMode_Changed 事件通知应用，应用程序可重新调用 GSXR_GetDegreeOfFreedom 获取新模式。

[基础能力规范] 设备跟踪模式变换事件

[功能实现]

- GSXR_EventType_DeviceState_TrackingMode_Changed 设备变换跟踪模式，XR Runtime 发送 GSXR_EventType_DeviceState_TrackingMode_Changed 事件进事件队列。

[注意]

若设备(通常为头显)跟踪模式变换同时造成空间原点重置时，除了 GSXR_EventType_DeviceState_TrackingMode_Changed 外，Runtime 亦须发送 GSXR_EventType_Common_SpaceOrigin_Relocalized 事件进入事件队列。

7.6.5. 跟踪系统边界

XR 跟踪系统的现实空间范围可能是有限的，范围内用户可以自由移动，范围外则跟踪系统可能无法维持跟踪或可能遭遇现实空间的障碍物。应用程序可能希望获取这些边界并更改应用程序行为或内容中的物件位置，以确保用户可以在边界范围内完成内容体验。

边界范围以空间原点为中心，可能为矩形、圆形、或不规则形，跟踪系统也可能提供边界防护墙提示用户。当空间原点发生变动时，边界将以新的空间原点描述相对位置，XR 应用程序必须重新获取以新原点为中心的边界位置。此外，系

统也有可能于应用程序运行期间允许用户更动空间边界，此时 XR 应用程序需要重新获取边界位置。

以下为三种空间边界可能发生变动的事件：

1. GSXR_EventType_Common_SpaceOrigin_Located 事件：应用程序重设空间原点类型。
2. GSXR_EventType_Common_SpaceOrigin_Relocalized 事件：跟踪系统重置空间原点。
3. GSXR_EventType_Common_SpaceBounds_Changed 事件：系统更动空间边界。

[基础能力规范] 空间边界更动事件

[功能实现]

- GSXR_EventType_Common_SpaceBounds_Changed 系统更动空间边界，XR Runtime 发送 GSXR_EventType_Common_SpaceBounds_Changed 事件进入事件队列。

以下介绍 GSXR 定义的空间边界相关函数。

[枚举] GSXR_SpaceBoundsVisible (空间边界防护墙可见模式)

名称	数值	描述
GSXR_SpaceBoundsVisible_System	0	交由系统决定边界防护墙显示模式
GSXR_SpaceBoundsVisible_ForceOn	1	强制显示边界防护墙

[枚举] GSXR_SpaceBoundsShape (空间边界形状)

名称	数值	描述
GSXR_SpaceBoundsShape_None	0	边界未设置
GSXR_SpaceBoundsShape_Rectangle	1	矩形边界

GSXR_SpaceBoundsShape_Circle	2	圆形边界
GSXR_SpaceBoundsShape_Irregular	3	不规则形边界

[结构体] GSXR_SpaceBoundsConfiguration (空间边界配置)

```
typedef struct GSXR_SpaceBoundsConfiguration {
    void*                       next;
    GSXR_Bool32                 isValid;
    GSXR_Bool32                 isEnabled;
    GSXR_SpaceBoundsVisible     visible;
    GSXR_SpaceBoundsShape       shape;
    uint32_t                    pointCount;
} GSXR_SpaceBoundsConfiguration;
```

说明：

空间边界配置结构体，用以描述边界信息

成员：

next 预留指针

isValid GSXR_TRUE 表示边界有效，GSXR_FALSE 表示边界无效

isEnabled GSXR_TRUE 表示边界系统开启，GSXR_FALSE 表示边界系统关闭

visible 边界防护墙可见模式

shape 边界形状

pointCount 边界形状若为不规则形，pointCount 为点集合数目；否则为 0

[函数] GSXR_GetSpaceBoundsConfiguration (获取空间边界配置信息)

```
GSXR_Result GSXR_GetSpaceBoundsConfiguration (
    GSXR_Runtime                        runtime,
    GSXR_SpaceBoundsConfiguration*      boundsConfiguration);
```

说明：

获取空间边界配置信息

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] boundsConfiguration 指向 GSXR_SpaceBoundsConfiguration 结构体的指针，

输出边界配置信息

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSpaceBoundsConfiguration

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `boundsConfiguration` 必须是指向 GSXR_SpaceBoundsConfiguration 结构的有效指针。

[函数实现]

- Runtime 获取当前边界配置信息填入 boundsConfiguration 输出。
- 若 Runtime 不支持边界系统，则返回 GSXR_Error_Function_Unsupported。

[返回值]

成功

- `GSXR_Result_Success` 空间边界配置获取成功。

失败

- `GSXR_Error_Operation_Failed` 空间边界配置获取失败。
- `GSXR_Error_Function_Unsupported` Runtime 不支持边界系统。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

根据获取的空间边界配置 GSXR_SpaceBoundsConfiguration 中的边界形状 GSXR_SpaceBoundsShape，可分别调用对应函数获取该边界形状的边界信息。

[函数] GSXR_GetSpaceBoundsRectangle（获取矩形边界信息）

```
GSXR_Result GSXR_GetSpaceBoundsRectangle (
      GSXR_Runtime          runtime,
      float*                width,
      float*                length);
```

说明：

获取矩形边界信息

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] width 输出矩形边界宽度，单位为米

[out] length 输出矩形边界长度，单位为米

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSpaceBoundsRectangle

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `width` 必须是指向 float 数值的有效指针。
- `length` 必须是指向 float 数值的有效指针。

[函数实现]

- Runtime 获取当前矩形边界的宽度及长度填入 width，length 输出。
- 若 Runtime 不支持矩形边界形状，则返回 GSXR_Error_Function_Unsupported。

[返回值]

成功

- `GSXR_Result_Success` 矩形边界获取成功。

失败

- `GSXR_Error_Operation_Failed` 矩形边界获取失败。

- `GSXR_Error_Function_Unsupported` Runtime 不支持矩形边界形状。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[函数] GSXR_GetSpaceBoundsCircle（获取圆形边界信息）

```
GSXR_Result GSXR_GetSpaceBoundsCircle (
    GSXR_Runtime            runtime,
    float*                  diameter);
```

说明：

获取圆形边界信息

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] diameter 输出圆形边界直径，单位为米

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSpaceBoundsCircle

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `diameter` 必须是指向 float 数值的有效指针。

[函数实现]

- Runtime 获取当前圆形边界的直径填入 diameter 输出。
- 若 Runtime 不支持圆形边界形状，则返回 GSXR_Error_Function_Unsupported。

[返回值]

成功

- GSXR_Result_Success 圆形边界获取成功。

失败

- GSXR_Error_Operation_Failed 圆形边界获取失败。
- GSXR_Error_Function_Unsupported Runtime 不支持圆形边界形状。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

[函数] GSXR_GetSpaceBoundsIrregular（获取不规则形边界信息）

```
GSXR_Result GSXR_GetSpaceBoundsIrregular (
      GSXR_Runtime              runtime,
      uint32_t                  pointCount,
      GSXR_Posef*               pointArray);
```

说明：

获取不规则形边界信息

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] pointCount 边界点集合数组个数，需相等于 GSXR_SpaceBoundsConfiguration 的 pointCount 数

[out] pointArray 指向 GSXR_Posef 结构体的数组，输出边界点集合位姿

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSpaceBoundsIrregular

[参数检验]

- runtime 必须是有效的 GSXR_Runtime 句柄。

- `pointCount` 相等于 GSXR_SpaceBoundsConfiguration 的 pointCount 数。
- `pointArray` 必须是指向 GSXR_Posef 数组的有效指针。

[函数实现]

- Runtime 获取当前不规则形边界的点集合位姿填入 pointArray 输出。
- 若 Runtime 不支持不规则形边界形状，则返回 GSXR_Error_Function_Unsupported。

[返回值]

成功

- `GSXR_Result_Success` 不规则形边界获取成功。

失败

- `GSXR_Error_Operation_Failed` 不规则形边界获取失败。
- `GSXR_Error_Function_Unsupported` Runtime 不支持不规则形边界形状。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

此外，若 Runtime 支持边界防护墙，GSXR 也提供了 XR 应用程序设置边界防护墙可见模式。

[函数] GSXR_SetSpaceBoundsVisible (设置边界防护墙可见模式)

```
GSXR_Result GSXR_SetSpaceBoundsVisible (
      GSXR_Runtime                 runtime,
      GSXR_SpaceBoundsVisible      visible);
```

说明：

设置边界防护墙可见模式

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] visible 指定边界防护墙可见模式

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_SetSpaceBoundsVisible

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `visible` 必须是有效的 GSXR_SpaceBoundsVisible 类型。

[函数实现]

- Runtime 根据 visible 决定该应用程序的边界防护墙可见模式。
- 若 Runtime 不支持边界防护墙，则返回 GSXR_Error_Function_Unsupported。

[返回值]

成功

- `GSXR_Result_Success` 边界防护墙可见模式设置成功。

失败

- `GSXR_Error_Operation_Failed` 边界防护墙可见模式设置失败。
- `GSXR_Error_Function_Unsupported` Runtime 不支持边界防护墙。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` visible 未定义于 GSXR_SpaceBoundsVisible 中。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

7.7. 设备信息

XR 应用程序可通过以下函数获取 XR 设备的相关信息，如：设备 SN 串号，firmware 版本号，电池电量等信息。

[结构体] GSXR_DeviceProperties (设备属性)

```
typedef struct GSXR_DeviceProperties {
   void*        next;
```

```
    char        serialNumber[GSXR_COMMON_STRING_MAX_SIZE];
    char        firmwareVersion[GSXR_COMMON_STRING_MAX_SIZE];
    char        vendorName[GSXR_COMMON_STRING_MAX_SIZE];
    char        algorithmVersion[GSXR_COMMON_STRING_MAX_SIZE];
    char        algorithmName[GSXR_COMMON_STRING_MAX_SIZE];
} GSXR_DeviceProperties;
```

说明：

设备属性结构体，用以描述设备的硬件及算法信息

成员：

next 预留指针

serialNumber 设备 SN 串号

firmwareVersion 设备 firmware 版本号

vendorName 设备厂家名称

algorithmVersion 设备算法版本号

algorithmName 设备算法名称

[函数] GSXR_GetDeviceProperties（获取设备属性）

```
GSXR_Result GSXR_GetDeviceProperties (
    GSXR_Runtime                runtime,
    GSXR_DeviceType             deviceType,
    GSXR_DeviceProperties*      deviceProperties);
```

说明：

获取设备属性

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[out] deviceProperties 指向 GSXR_DeviceProperties 结构体的指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetDeviceProperties

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `deviceProperties` 必须是指向 GSXR_DeviceProperties 结构的有效指针。

[函数实现]

- Runtime 获取 deviceType 的设备属性信息填入 deviceProperties 中输出。

[返回值]

成功

- `GSXR_Result_Success` 设备属性信息获取成功。

失败

- `GSXR_Error_Operation_Failed` 设备属性信息获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 deviceProperties 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[结构体] GSXR_BatteryStatus (设备电池状态)

```
typedef struct GSXR_BatteryStatus {
   void*            next;
   float            percentange;
   GSXR_Bool32      charging;
```

```
} GSXR_BatteryStatus;
```

说明：

设备电池状态结构体，用以描述设备的电池电量及充电状态

成员：

next 预留指针

percentange 电池剩余电量百分比

charging GSXR_TRUE 表示电池处于充电状态，GSXR_FALSE 表示电池未处于充电状态

[函数] GSXR_GetBatteryStatus（获取设备电池状态）

```
GSXR_Result GSXR_GetBatteryStatus (
    GSXR_Runtime                runtime,
    GSXR_DeviceType             deviceType,
    GSXR_BatteryStatus*         batteryStatus);
```

说明：

获取设备电池状态

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[out] batteryStatus 指向 GSXR_BatteryStatus 结构体的指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetBatteryStatus

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `deviceType` 必须是有效的 GSXR_DeviceType 类型。
- `batteryStatus` 必须是指向 GSXR_BatteryStatus 结构的有效指针。

[函数实现]

- Runtime 获取 deviceType 的电池状态填入 batteryStatus 中输出。

[返回值]

成功

- `GSXR_Result_Success` 设备电池状态获取成功。

失败

- `GSXR_Error_Operation_Failed` 设备电池状态获取失败。
- `GSXR_Error_Function_Unsupported` deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- `GSXR_Error_Device_Disconnected` deviceType 尚未连线。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` deviceType 未定义于 GSXR_DeviceType 中。或 batteryStatus 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

8. DM 上报接口说明

8.1. 结构体

8.1.1. 时间

[结构体] GSXR_TimeS (时间)

```
typedef struct GSXR_TimeS {
    long            time;
    uint32_t        year;
    uint32_t        month;
    uint32_t        day;
    uint32_t        hour;
    uint32_t        minute;
    uint32_t        second;
```

```
} GSXR_TimeS;
```

说明：

时间结构体

成员：

Time 时间精确到毫秒

Year 年

Month 月

Day 日

Hour 时

Minute 分

Second 秒

8.1.2. 经纬度

[结构体] GSXR_Device_Long_Lat (经纬度)

```
typedef struct GSXR_Device_Long_Lat {
    float           longitude;
    float           latitude;
} GSXR_Device_Long_Lat;
```

说明：

经纬度

成员：

longitude 经度

latitude 纬度

8.1.3. 码表信息 业务埋点信息

[结构体] GSXR_Buried_Point_Info (码表信息 业务埋点信息)

```
typedef struct GSXR_Buried_Point_Info {
    Char             stopwatch[GSXR_COMMON_STRING_MAX_SIZE];
    char             buried_point[GSXR_COMMON_STRING_MAX_SIZE];
} GSXR_Buried_Point_Info;
```

说明：

码表信息 业务埋点信息

成员：

stopwatch 码表信息

buried_point 埋点信息

8.1.4. 采集信息

[结构体] GSXR_Timing_Gather_Info (采集信息)

```
typedef struct GSXR_Timing_Gather_Info{
    float           value;
    gsxr_time_t     now_time;
}GSXR_Timing_Gather_Info;
```

说明：

采集信息

成员：

value 取值 cpu 温度、ROM 使用率、剩余电量、屏幕亮度

now_time 当前时间

8.1.5. 充电信息

[结构体] GSXR_Top_Electricity_Info (充电信息)

```
typedef struct GSXR_Top_Electricity_Info {
    gsxr_time_t     start;
    gsxr_time_t     end;
    float           start_temperature;
    Float           end_temperature;
    float           start_voltage;
    float           end_voltage;
}GSXR_Top_Electricity_Info;
```

说明：

充电信息结构体

成员：

Start 开始充电时间

End 结束充电时间

start_temperature 开始充电温度
end_temperature 结束充电温度
start_voltage 开始充电电压
end_voltage 结束充电电压

8.1.6. 引用信息

[结构体] GSXR_App_Info (应用信息)

```
typedef struct GSXR_App_Info {
    Char            app_name[GSXR_COMMON_STRING_MAX_SIZE];
    Char            package_name[GSXR_COMMON_STRING_MAX_SIZE];
    Long            begin_time_len;
    Long            end_time_len;
} GSXR_App_Info;
```

说明:

应用信息

成员:

app_name 应用名称
package_name 包名
begin_time_len; 启动时间以毫秒表示的 Unix 时间
end_time_len; 关闭时间以毫秒表示的 Unix 时间

8.1.7. DM 数据结构体使用信息

[结构体] GSXR_Dm_Reported_Use_Info (DM 数据结构体使用信息)

```
typedef struct GSXR_Dm_Reported_Use_Info {
    gsxr_app_info_t             app_info;
    uint32_t                    location_mode;
    gsxr_device_long_lat_t      long_lat;
    gsxr_Buried_point_info_t    Buried_point_info;
    gsxr_timing_gather_info_t   start_screen;
```

```
    gsxr_timing_gather_info_t    end_screen;
    gsxr_timing_gather_info_t    start_luminance_info;
    gsxr_timing_gather_info_t    end_luminance_info;
    gsxr_timing_gather_info_t    start_volume_info;
    gsxr_timing_gather_info_t    end_volume_info;
    gsxr_top_electricity_info_t  electricity_info;
}GSXR_Dm_Reported_Use_Info;
```

说明：

设备电池状态结构体

成员：

app_info; 设备上各应用启动次数及时长

location_mode; 定位模式 gsxr_app_use_info_list_t 的位掩码

long_lat; 经纬度

Buried_point_info; 码表信息和业务埋点信息

start_screen; 点亮屏幕开始信息

end_screen; 点亮屏幕结束信息

start_luminance_info; 亮度开始调节时间及亮度值

end_luminance_info; 亮度结束调节时间及亮度值

start_volume_info; 音量开始调节时间及音量值

end_volume_info; 音量结束调节时间及音量值

electricity_info; 采集开始充电时间,结束充电时间,开始充电温度,结束充电温度开始充电电压,结束充电电压开始充电|电池状态,结束充电电池状态,本次充电时长

8.1.8. 分辨率

[结构体] GSXR_Resolution_Info (分辨率)

```
typedef struct GSXR_Resolution_Info {
    uint32_t        width;
    uint32_t        height;
} GSXR_Resolution_Info;
```

说明：

分辨率结构体

成员：

Width 屏幕宽度

Height 屏幕高度

8.1.9. 设备所使用的操作系统及版本号

[结构体] GSXR_Operation_Info（设备所使用的操作系统及版本号）

```
typedef struct GSXR_Operation_Info {
    Char            operation_system[GSXR_COMMON_STRING_MAX_SIZE];
    Char            version_number[GSXR_COMMON_STRING_MAX_SIZE];
}GSXR_Operation_Info;
```

说明：

设备所使用的操作系统及版本号

成员：

operation_system 操作系统

version_number 版本号

8.1.10. 设备所使用的固件名称及版本号

[结构体] GSXR_Firmware_Info（设备所使用的固件名称及版本号）

```
typedef struct GSXR_Firmware_Info {
    Char            firmware[GSXR_COMMON_STRING_MAX_SIZE];
    Char            version_number[GSXR_COMMON_STRING_MAX_SIZE];
}GSXR_Firmware_Info;
```

说明：

设备所使用的固件名称及版本号

成员：

firmware 固件名称

version_number 版本号

8.1.11. 厂商信息

[结构体] GSXR_Manufacturer_Info (厂商信息)

```
typedef struct GSXR_Manufacturer_Info {
    Char            name[GSXR_COMMON_STRING_MAX_SIZE];
    Char            site[GSXR_COMMON_STRING_MAX_SIZE];
    Char            http[GSXR_COMMON_STRING_MAX_SIZE];
    Char            phone[GSXR_COMMON_STRING_MAX_SIZE];
}GSXR_Manufacturer_Info;
```

说明：

厂商信息结构体

成员：

Name	厂商名称
Site	厂商地址
http	厂商官网
phone	厂商电话

8.1.12. 设备 CPU/GPU 信息

[结构体] GSXR_Device_CPU_GPU_Info (设备 CPU/GPU 信息)

```
typedef struct GSXR_Device_CPU_GPU_Info{
    Char            name[GSXR_COMMON_STRING_MAX_SIZE];
    Int             kernel_num;
}GSXR_Device_CPU_GPU_Info;
```

说明：

设备 CPU/GPU 信息结构体

成员：

Name	型号名称
kernel_num	核心数

8.1.13. DM 数据结构体基础信息

[结构体] GSXR_DM_Reported_Info (DM 数据结构体基础信息)

```
typedef struct GSXR_DM_Reported_Info {
    Char                        sole_mark[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        device_type[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        algorithm_version[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        platform[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        versions_name[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        model[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        sdk_version[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        ip_site[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        mac_site[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        rom[GSXR_COMMON_STRING_MAX_SIZE];
    Char                        ram[GSXR_COMMON_STRING_MAX_SIZE];
    gsxr_resolution_info_t      resolution;
    gsxr_time_t                 create_time;
    gsxr_operation_info_t       operation_info;
    gsxr_firmware_info_t        firmware_info;
    gsxr_manufacturer_info_t    manufacturer_info;
    gsxr_device_cpu_gpu_info_t  cpu_info;
    gsxr_device_cpu_gpu_info_t  gpu_info;
} GSXR_DM_Reported_Info;
```

说明：

DM 数据结构体基础信息

成员：

sole_mark	设备唯一号
device_type	设备型号名
algorithm_version	厂商标识
Platform	设备所使用的硬件芯片平台版本
versions_name	主版本名称
model	主版本名称型号
sdk_version	设备所使用的的 SDK 版本号
ip_site	设备 ip 地址
mac_site	设备 MAC 地址
rom	设备 ROM 信息(单位 kb)

ram	设备 RAM 信息(单位 kb)
resolution	屏幕分辨率
create_time	设备创建时间
operation_info	设备操作系统及版本号信息
firmware_info	设备所使用的固件名称及版本号
manufacturer_info	厂商信息
cpu_info	设备 CPU 信息
gpu_info	设备 GPU 信息

8.2. DM 接口

8.2.1. DM 指定数据类型获取

[枚举] GSXR_Timing_Gather (采集信息类型)

名称	数值	描述
CPU_TEMPERATURE	1	cpu 温度
ROM_USAGE_RATE	2	ROM 使用率
ELECTRICITY_QUANTIT	4	剩余电量

[函数] GSXR_GetDMData (获取指定设备 DM 数据)

```
extern GSXR_EXPORT GSXR_Result GSXR_GetDMData(
    GSXR_Runtime                    runtime,
    GSXR_DeviceType                 deviceType,
    GSXR_Timing_Gather              timing_gather_Type,

    GSXR_Timing_Gather_Info*        timing_gather);
```

说明:

获取指定设备 DM 数据

参数:

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] deviceType 指定设备类型

[in] timing_gather_Type 为 GSXR_Timing_Gather 的掩码指定请求数据类型

[out] batteryStatus 指向 GSXR_DM_Reported_Info 结构体之指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetDMData

[参数检验]

- runtime 必须是有效的设备句柄。
- deviceType 必须是有效的设备类型。
- timing_gather_Type 为 GSXR_Timing_Gather 的值，指定要获取的数据类型
- battery_status* 指向 GSXR_DM_Reported_Info 结构体，填入获取的数据

[函数实现]

- 获取 DM 信息 deviceType 指定设备类型 timing_gather_Type 指定数据类型，数据填入 battery_status 中返回。

[返回值]

成功

- GSXR_Result_Success DM 数据获取成功。

失败

- GSXR_Error_Operation_Failed DM 数据获取失败。
- GSXR_Error_Function_Unsupported deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- GSXR_Error_Device_Disconnected deviceType 尚未连线。

- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceType 未定义于 GSXR_DeviceType 中。或 specifiedTime 为一无效数据。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态

8.2.2. 设置 DM 运行时信息回调

[类型定义] GSXR_DmEventCallbackFn (设备 DM 事件回调函数)

```
typedef void (*GSXR_DmEventCallbackFn) (
      GSXR_Dm_Reported_Use_Info dm_use);
```

说明：

设备 DM 信息事件回调函数，当设备触发相关 DM 使用信息时回调

参数：

[in] dm_use 指向 GSXR_Dm_Reported_Use_Info 结构体参数

返回值：

无

备注：

dm_use 指向结构体只填写所发生事件相关参数即可其他无关参数需置零

[函数]GSXR_SetDmEventCallback(设置 DM 运行时事件回调函数)

```
extern GSXR_EXPORT GSXR_Result GSXR_SetDmEventCallback(
       GSXR_Runtime                    runtime,
     GSXR_DmEventCallbackFn            Callback）;
```

说明：

设置事件回调函数，各设备于指定 DM 数据事件发生时回调

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] callback 事件回调函数指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

[基础能力规范] GSXR_SetDmEventCallback

[参数检验]

- Runtime 必须是有效的设备句柄。
- callback 事件回调函数指针。

[函数实现]

- 设置 DM 运行时事件回调函数，于指定 DM 数据事件发生时调用回调并将数据填入回调参数 dm_use 中，只需填写发生事件相关参数即可，其余参数需置零。

[返回值]

成功

- GSXR_Result_Success DM 数据获取成功。

失败

- GSXR_Error_Operation_Failed DM 数据获取失败。
- GSXR_Error_Function_Unsupported deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- GSXR_Error_Device_Disconnected deviceType 尚未连线。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceType 未定义于 GSXR_DeviceType 中。或 specifiedTime 为一无效数据。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态

8.2.3. 获取设备 DM 数据结构基础信息

[函数] GSXR_DMReportedInfo (获取设备 DM 数据结构基础信息)

```
extern GSXR_EXPORT GSXR_Result GSXR_DMReportedInfo(
    GSXR_Runtime                    runtime,
    GSXR_DM_Reported_Info*          reported);
```

说明：

获取设备 DM 数据结构基础信息

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] reported* 指向 GSXR_DM_Reported_Info 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Device_Disconnected

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

[基础能力规范] GSXR_DMReportedInfo

[参数检验]

- Runtime 必须是有效的设备句柄。
- reported* 指向 GSXR_DM_Reported_Info 结构体。

[函数实现]

- 获取设备 DM 数据结构基础信息，填入到 reported 中。

[返回值]

成功

- `GSXR_Result_Success` DM 数据获取成功。

失败

- GSXR_Error_Operation_Failed DM 数据获取失败。
- GSXR_Error_Function_Unsupported deviceType 定义于 GSXR_DeviceType 中，但 Runtime 不支持该设备。
- GSXR_Error_Device_Disconnected deviceType 尚未连线。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid deviceType 未定义于 GSXR_DeviceType 中。或 specifiedTime 为一无效数据。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态

9. XR 渲染器函数说明

XR 应用程序从 XR 设备获取了输入部件数据及空间位姿数据后，必须将这些输入数据转化为应用内容的可用数据，运用于应用本身的内容逻辑，最终将内容渲染至帧缓冲中，输出显示屏进行终端显示。至此，用户才真正可从 XR 显示设备中获取 XR 的即时视觉交互。

一个标准的 XR 渲染器，必关联以下基础元素。以下描述是各元素基本概念：

1. 图形 API

无论是 XR 应用程序本身的内容渲染，还是 XR Runtime 对于应用提交的内容帧所进行的 XR 渲染管线后处理渲染，都应该使用操作系统所支持的图形 API 进行渲染。故应用程序在 XR 渲染器创建之初，必须将应用程序使用的图形 API 特化信息提供给 Runtime，XR 渲染器才能正确进行后续的 XR 后处理渲染。

2. 视图配置

一个标准的 XR 显示设备，可以是穿戴于头上的头戴式显示器，也可能是握持于手上的手持式设备，不同性质的显示设备所需求的渲染视图也可能不同。通常头戴式显示器需求渲染左右两眼的双视图，而手持式设备则只需求渲染单视图。同样地，显示设备的规格差异将产生不同的视图配置，也将影响 XR Runtime 渲染器在建构渲染循环时的帧缓冲操作。

3. 纹理队列

为了获取 XR 应用程序的渲染内容，XR Runtime 提供纹理队列图像予 XR 应用程序进行渲染。应用程序可按照 XR Runtime 支持的纹理格式及参数，根据自身的渲染需求创建纹理队列，并于渲染循环中循序从队列中获取可用纹理进行内容渲染。

4. 渲染循环

XR 应用程序在备妥所有渲染必要要件后，便可开始进行渲染循环。一个标准的渲染循环需先等待 Runtime 返回 V-Sync 信号，在轮询事件队列、获取输入及位姿数据后，便可从纹理队列中获取纹理进行内容渲染后提交该纹理图像给 Runtime 进行后续的 XR 后处理渲染，如此反复循环便为渲染循环。

本章后续章节将逐步详述 GSXR 对于 XR 渲染器各元素的基础设计及相关函数操作。

9.1. XR 渲染器的生命周期

XR 应用程序必须先完成 XR 渲染器的实例创建及初始，才可以提供 XR 应用程序后续渲染循环所需求的功能。在 XR 应用程序不再使用此渲染器时，应用程序必须停止 XR 渲染器并销毁实例，将资源返回系统，此流程即是 XR 渲染器的生

命周期。

```
GSXR_DEFINE_HANDLE(GSXR_Renderer)
```

9.1.1. 创建及初始渲染器实例

GSXR 提供 GSXR_CreateRenderer 函数进行渲染器实例的创建及初始。

[枚举] GSXR_GraphicsApi (图形 API 类型)

名称	数值	描述
GSXR_GraphicsApi_OpenGLES	0	OpenGL ES
GSXR_GraphicsApi_OpenGL	1	OpenGL

[结构体] GSXR_RendererCreateInfo (渲染器初始信息)

```
typedef struct GSXR_RendererCreateInfo {
   void*                 next;
   GSXR_GraphicsApi      graphicsApi;
   GSXR_DisplayId        displayId;
} GSXR_RendererCreateInfo;
```

说明:

渲染器初始信息结构体, 用以描述创建渲染器需求的初始信息

成员:

next 预留指针

graphicsApi 图形 API 类型

displayId GSXR_DisplayId 标识号, 由 GSXR_GetDisplay 获取

[函数] GSXR_CreateRenderer (创建渲染器)

```
GSXR_Result GSXR_CreateRenderer (
      GSXR_Runtime                              runtime,
      const GSXR_RendererCreateInfo*            createInfo,
      GSXR_Renderer*                            renderer);
```

说明：

创建渲染器

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] createInfo 指向 GSXR_RendererCreateInfo 结构体，内含渲染器初始信息

[out] renderer 指向 GSXR_Renderer 句柄的指针，渲染器创建成功输出一有效句柄，创建失败输出 GSXR_NULL_HANDLE

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_CreateRenderer

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `createInfo` 必须是指向 GSXR_RendererCreateInfo 结构的有效指针。
- `renderer` 必须是指向 GSXR_Renderer 句柄的有效指针。

[函数实现]

- Runtime 根据 createInfo 输入的图形 API 类型及其特化信息完成渲染器初始化，并输出有效的 renderer 句柄。若创建失败则输出 GSXR_NULL_HANDLE。
- 渲染器初始化后，Runtime 发送 GSXR_EventType_RendererState_Renderer_Created 事件进事件队列，GSXR_RendererStateEvent 的成员 renderer 为创建的渲染器句柄，

viewType 为 GSXR_ViewType_None，textureQueue 及 looper 为 GSXR_NULL_HANDLE。

[返回值]

成功

- GSXR_Result_Success 渲染器创建成功。

失败

- GSXR_Error_Operation_Failed 渲染器创建失败。
- GSXR_Error_Function_Unsupported graphicsApi 定义于 GSXR_GraphicsApi 中，但 Runtime 不支持该图形 API。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid createInfo 为 nullptr 或检验为无效的输入参数。或 renderer 为 nullptr。
- GSXR_Error_Out_Of_Memory 内存溢出。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

9.1.2. 终止及销毁渲染器实例

GSXR 提供 GSXR_DestroyRenderer 函数进行渲染器实例的终止及销毁。

[函数] GSXR_DestroyRenderer (销毁渲染器)

```
GSXR_Result GSXR_DestroyRenderer (
    GSXR_Renderer                          renderer);
```

说明：

销毁渲染器

参数：

[in] renderer GSXR_Renderer 句柄，由 GSXR_CreateRenderer 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_DestroyRenderer

[参数检验]

- `renderer` 必须是有效的 GSXR_Renderer 句柄。

[函数实现]

- Runtime 根据输入的 renderer 句柄终止并销毁渲染器。
- 渲染器终止后，Runtime 发送 GSXR_EventType_RendererState_Renderer_Destroyed 事件进事件队列，GSXR_RendererStateEvent 的成员 renderer 为销毁的渲染器句柄，viewType 为 GSXR_ViewType_None，textureQueue 及 looper 为 GSXR_NULL_HANDLE。

[返回值]

成功

- `GSXR_Result_Success` 渲染器销毁成功。

失败

- `GSXR_Error_Operation_Failed` 渲染器销毁失败。
- `GSXR_Error_Handle_Invalid` renderer 为无效句柄。
- `GSXR_Error_Call_Flow_Invalid` 调用此函数前未先行调用 GSXR_CreateRenderer。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

9.2. 视图集配置

视图集为视图的集合体，其中可能包含一个或多个视图，同一视图集中的各视图配置全部相同。一个视图集通常实际对应至一组显示输出（可能为实体显示设备，如 VR 头显；也可能为虚拟显示设备，如无线投屏），Runtime 经由视图集配置描述显示设备的组态及其对应的纹理结构，XR 应用程序需要根据视图集配置指定其图形 API 相关参数进行渲染，才可以渲染出合适的图像并输出到对应此视图集配置的显示设备。

显示设备具有主屏、副屏的分，一般 XR 应用程序只以主屏为渲染对象（且多数情况也只存在主屏），主屏对应至创建渲染器时所输入由显示设备获取的 GSXR_DisplayId，此 GSXR_DisplayId 从 7.3 节所介绍的函数 GSXR_GetDisplay 经由 GSXR_DeviceType 获取而来，而后用于渲染主屏视图的视图姿态即从此 GSXR_DeviceType 的设备姿态转置而来。副屏通常为主屏的延伸屏幕，使用与主屏相同的设备姿态并根据副屏视图配置依需求进行副屏渲染。在某些特定的应用情境下，同一渲染器可能同时存在主屏及副屏，XR 应用程序可根据应用本身设计决定是否同时渲染主、副屏，则至少可以会渲染主屏。

[枚举] GSXR_ViewType (视图类型)

名称	数值	描述
GSXR_ViewType_None	0	无效的视图
GSXR_ViewType_Primary_Mono	1	主屏单视图
GSXR_ViewType_Primary_Stereo	2	主屏双视图
GSXR_ViewType_Secondary_Mono	17	副屏单视图
GSXR_ViewType_Secondary_Stereo	18	副屏双视图

[结构体] GSXR_ViewSetConfiguration (视图集配置)

```
typedef struct GSXR_ViewSetConfiguration {
    void*                   next;
    GSXR_ViewType           viewType;
    uint32_t                viewCount;
    uint32_t                textureFormatCount;
    uint32_t                targetWidth;
    uint32_t                maxWidth;
    uint32_t                targetHeight;
    uint32_t                maxHeight;
```

```
    uint32_t                targetSampleCount;
    uint32_t                maxSampleCount;
    uint32_t                fps;
    uint32_t                maxLayerCount;
    uint32_t                Ipd;
    float                   fovHRad;
    float                   fovVRad;
} GSXR_ViewSetConfiguration;
```

说明：

视图集配置结构体，用以描述对应于显示设备的视图参数

成员：

next 预留指针

viewType 视图类型（同一渲染器下的各视图集配置 viewType 不重复）

viewCount 视图集的视图数目

textureFormatCount 纹理队列支持的格式数目

targetWidth 纹理图像目标宽度

maxWidth 纹理图像最大宽度

targetHeight 纹理图像目标高度

maxHeight 纹理图像最大高度

targetSampleCount 纹理图像目标采样数

maxSampleCount 纹理图像最大采样数

fps 渲染帧率

maxLayerCount 渲染合成层最大层数

ipd 瞳距

fovHRad 水平 FOV

fovVRad 垂直 Fov

当同时存在主、副屏时，Runtime 必须使用两个视图集，配置各自描述的主、副屏视图信息。例如：当前可能连接两款显示屏设备，一为 VR 头戴显示器，另一为无线投屏，头显（用户穿戴）需采用分屏双视图与畸变校正，投屏（分享观众）则采用全屏单视图且无需畸变校正，主屏、副屏需求的纹理图像大小、渲染帧率等设置也可能不同。在此使用情境，渲染器具有两份视图配置（VR 头显的主

视图配置 viewType 为 GSXR_ViewType_Primary_Stereo，viewCount 为 2，纹理图像大小为单屏的宽高，fps 为 75。无线投屏的副视图配置 viewType 为 GSXR_ViewType_Secondary_Mono，viewCount 为 1，纹理图像大小为全屏的宽高，fps 为 30。），应用程序需根据主副视图配置生成对应的纹理队列同时进行两个渲染循环。

[函数] GSXR_GetViewSetCount（获取渲染器的视图集数目）

```
GSXR_Result GSXR_GetViewSetCount (
    GSXR_Runtime          runtime,
    uint32_t*             count);
```

说明：

获取渲染器的视图集数目

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] count 视图集数目

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetViewSetCount

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `count` 必须是指向 uint32_t 数值的有效指针。

[函数实现]

- 获取视图集数目填入 count 输出。

[返回值]

成功

- `GSXR_Result_Success` 视图集数目获取成功。

失败

- `GSXR_Error_Operation_Failed` 视图集数目获取失败。
- `GSXR_Error_Handle_Invalid` tuntime 为无效句柄。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

[函数] GSXR_GetViewSetConfigurations (获取渲染器的视图集配置)

```
GSXR_Result GSXR_GetViewSetConfigurations (
    GSXR_Runtime                    runtime,
    uint32_t                        count,
    GSXR_ViewSetConfiguration*      configurations);
```

说明：

获取渲染器的视图集配置

参数：

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] count configurations 数组个数, 需相等于 GSXR_GetViewSetCount 的 count 数

[out] configurations 指向 GSXR_ViewSetConfiguration 结构体的数组

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetViewSetConfigurations

[参数检验]

- runtime 必须是有效的 GSXR_Runtime 句柄。
- count 必须相等于 GSXR_GetViewSetCount 的 count 数。
- configurations 必须是指向 GSXR_ViewSetConfiguration 数组的有效指针，数组个数为 count。

[函数实现]

- 获取视图集配置资讯填入 configurations 输出。

[返回值]

成功

- GSXR_Result_Success 视图集配置获取成功。

失败

- GSXR_Error_Operation_Failed 视图集配置获取失败。
- GSXR_Error_Handle_Invalid renderer 为无效句柄。
- GSXR_Error_Argument_Invalid count 不等于 GSXR_GetViewSetCount 的 count 数。或 configurations 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Renderer_Unavailable 渲染器未处于正确工作状态。

Runtime 运行期间，由于显示设备连线状态或显示屏设置的变化，渲染器的视图配置会随之改变。Runtime 必须于视图配置发生变动时发送以下事件通知应用程序，应用程序必须按照事件类型调整对应的渲染处理。

- GSXR_EventType_RendererState_ViewSet_Changed（视图集内容变更）
- GSXR_EventType_RendererState_ViewSet_Added（视图集添加）
- GSXR_EventType_RendererState_ViewSet_Removed（视图集删除）

[基础能力规范] 视图集事件

[功能实现]

- GSXR_EventType_RendererState_ViewSet_Changed 渲染器运行期间，现有视图集配置发

生变动，XR Runtime 发送 GSXR_EventType_RendererState_ViewSet_Changed 事件进事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄，viewType 为变动视图集的视图类型，textureQueue 及 looper 为 GSXR_NULL_HANDLE。

- GSXR_EventType_RendererState_ViewSet_Added 渲染器运行期间，添加了新视图集，XR Runtime 发送 GSXR_EventType_RendererState_ViewSet_Added 事件进入事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄，viewType 为添加视图集的视图类型，textureQueue 及 looper 为 GSXR_NULL_HANDLE。
- GSXR_EventType_RendererState_ViewSet_Removed 渲染器运行期间，删除了现有视图集，XR Runtime 发送 GSXR_EventType_RendererState_ViewSet_Removed 事件进入事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄，viewType 为删除视图集的视图类型，textureQueue 及 looper 为 GSXR_NULL_HANDLE。

9.3. 纹理队列

应用程序要将内容图像呈现至显示设备，必须先将图像渲染至帧缓冲中。为使 XR Runtime 有效管理 XR 应用程序的渲染图像以进行 XR 渲染管线的渲染后处理，Runtime 提供了纹理队列机制，应用程序可从纹理队列中获取可用纹理进行渲染，且 Runtime 必须允许应用程序创建多个纹理队列并同时使用。

9.3.1. 纹理队列的创建与销毁

纹理队列的创建必须由 XR 应用程序设置其渲染图像所需参数，包含启用的功能选项、纹理图像格式、纹理大小、采样数、… 等，这些纹理参数关联操作系统所使用的图形 API，XR Runtime 必须根据图形 API 支持能力及 XR 应用程序输入的纹理图像参数创建纹理队列，供 XR 应用程序于渲染循环中获取纹理渲染使用。XR 应用程序不再使用该纹理队列必须对于该纹理队列进行销毁，将资源返还系统。

纹理队列功能选项为 XR Runtime 提供给 XR 应用程序选用的纹理功能，XR 应用程序可经由 GSXR_GetTextureQueueOptions 函数获取 XR Runtime 所支持的纹理功能选项。

[枚举] GSXR_TextureQueueOptions (纹理队列功能选项)

名称	数值	描述
GSXR_TextureQueueOptions_Protected_Content	0x00000001	纹理内存受保护
GSXR_TextureQueueOptions_Color_Attachment	0x00010000	纹理附加颜色属性
GSXR_TextureQueueOptions_Manual_Management	0x10000000	应用自建纹理队列

[函数] GSXR_GetTextureQueueOptions (获取纹理队列功能选项)

```
GSXR_Result GSXR_GetTextureQueueOptions (
    GSXR_Renderer            renderer,
    GSXR_Flags64*            recommendedOptions,
    GSXR_Flags64*            supportedOptions);
```

说明：

获取纹理队列功能选项, 包含建议的及所有的支持选项

参数：

[in] renderer GSXR_Renderer 句柄, 由 GSXR_CreateRenderer 获取

[out] recommendedOptions GSXR_TextureQueueOptions 的位掩码, 输出系统建议的纹理队列功能选项

[out] supportedOptions GSXR_TextureQueueOptions 的位掩码, 输出可支持的所有纹理队列功能选项

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetTextureQueueOptions

[参数检验]

- renderer 必须是有效的 GSXR_Renderer 句柄。
- recommendedOptions 必须是指向 GSXR_Flags64 数值的有效指针。
- supportedOptions 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- 渲染器 renderer 获取纹理队列功能选项，将建议的以及所有的支持功能选项各自进行 OR 运算后填入 recommendedOptions 及 supportedOptions 输出。

[返回值]

成功

- GSXR_Result_Success 纹理队列功能选项获取成功。

失败

- GSXR_Error_Operation_Failed 纹理队列功能选项获取失败。
- GSXR_Error_Handle_Invalid renderer 为无效句柄。
- GSXR_Error_Argument_Invalid recommendedOptions 为 nullptr。或 supportedOptions 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Renderer_Unavailable 渲染器未处于正确工作状态。

纹理图像格式通常为各图形 API 自定义的数值，在创建纹理队列之前，XR 应用程序可经由 GSXR_GetSupportedTextureFormats 函数获取 XR Runtime 所支持的纹理图像格式再决定创建何种图像格式的纹理队列。

```
typedef int64_t GSXR_TextureFormat;
```

[函数] GSXR_GetSupportedTextureFormats (获取纹理队列的纹理格式)

```
GSXR_Result GSXR_GetSupportedTextureFormats (
    GSXR_Renderer              renderer,
    uint32_t                   formatCount,
    GSXR_TextureFormat*        formats);
```

说明：

获取纹理队列支持的纹理格式

参数：

[in] renderer GSXR_Renderer 句柄，由 GSXR_CreateRenderer 获取

[in] formatCount formats 数组个数，需相等于 GSXR_ViewSetConfiguration 的 textureFormatCount 数

[out] formats 指向 GSXR_TextureFormat 的数组

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetSupportedTextureFormats

[参数检验]

- `renderer` 必须是有效的 GSXR_Renderer 句柄。
- `formatCount` 必须相等于 GSXR_ViewSetConfiguration 的 textureFormatCount 数。
- `formats` 必须是指向 GSXR_TextureFormat 数组的有效指针，数组个数为 formatCount。

[函数实现]

- 渲染器 renderer 获取支持的纹理格式填入 formats 输出。

[返回值]

成功

- `GSXR_Result_Success` 纹理格式获取成功。

失败

- `GSXR_Error_Operation_Failed` 纹理格式获取失败。
- `GSXR_Error_Handle_Invalid` renderer 为无效句柄。
- `GSXR_Error_Argument_Invalid` formatCount 不等于 GSXR_ViewSetConfiguration 的 textureFormatCount。或 formats 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

[结构体] GSXR_TextureQueueCreateInfo (纹理队列初始资讯)

```
typedef struct GSXR_TextureQueueCreateInfo {
    void*                   next;
    GSXR_Flags64            options;
    GSXR_TextureFormat      format;
    uint32_t                width;
    uint32_t                height;
    uint32_t                sampleCount;
    uint32_t                faceCount;
    uint32_t                mipLevelCount;
    uint32_t                arrayLayerCount;
} GSXR_TextureQueueCreateInfo;
```

说明：

纹理队列初始资讯结构体，用以描述纹理参数

成员：

next 预留指针

options 设置启用的纹理功能选项

format 纹理图像格式，数值为各图形 API 自定义

width 纹理图像宽度

height 纹理图像高度

sampleCount 纹理图像采样数

faceCount 纹理图像张数

mipLevelCount 纹理图像 mipmap 层级数

arrayLayerCount 纹理图像数组阶层数

纹理队列及队列中的纹理在创建完成后全部都以句柄识别，句柄定义如下：

```
GSXR_DEFINE_HANDLE(GSXR_TextureQueue)
GSXR_DEFINE_HANDLE(GSXR_Texture)
```

[函数] GSXR_CreateTextureQueue （创建纹理队列）

```
GSXR_Result GSXR_CreateTextureQueue (
    GSXR_Renderer                                renderer,
    const GSXR_TextureQueueCreateInfo*           createInfo,
    GSXR_TextureQueue*                           textureQueue);
```

说明：

创建纹理队列

参数：

[in] renderer GSXR_Renderer 句柄，由 GSXR_CreateRenderer 获取

[in] createInfo 指向 GSXR_TextureQueueCreateInfo 结构体

[out] textureQueue 纹理队列创建成功输出一个有效句柄，创建失败输出 GSXR_NULL_HANDLE

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_CreateTextureQueue

[参数检验]

- `renderer` 必须是有效的 GSXR_Renderer 句柄。

- `createInfo` 必须是指向 GSXR_TextureQueueCreateInfo 结构的有效指针。
- `textureQueue` 必须是指向 GSXR_TextureQueue 数值的有效指针。

[函数实现]

- 渲染器 renderer 根据 createInfo 资讯创建纹理队列，并将句柄填入 textureQueue 输出。
- 完成纹理队列创建后，发送 GSXR_EventType_RendererState_TextureQueue_Created 事件进事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄，viewType 为 GSXR_ViewType_None，textureQueue 为创建的纹理队列句柄，looper 为 GSXR_NULL_HANDLE。

[返回值]

成功

- `GSXR_Result_Success` 纹理队列创建成功。

失败

- `GSXR_Error_Operation_Failed` 纹理队列创建失败。
- `GSXR_Error_Handle_Invalid` renderer 为无效句柄。
- `GSXR_Error_Argument_Invalid` createInfo 为 nullptr 或检验为无效的输入参数。或 textureQueue 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

[函数] GSXR_DestroyTextureQueue (销毁纹理队列)

```
GSXR_Result GSXR_DestroyTextureQueue (
    GSXR_TextureQueue                    textureQueue);
```

说明：

销毁纹理队列

参数：

[in] textureQueue GSXR_TextureQueue 句柄，由 GSXR_CreateTextureQueue 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_DestroyTextureQueue

[参数检验]

- `textureQueue` 必须是有效的 GSXR_TextureQueue 句柄。

[函数实现]

- 渲染器销毁纹理队列 textureQueue。
- 纹理队列销毁后，发送 GSXR_EventType_RendererState_TextureQueue_Destroyed 事件进事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄，viewType 为 GSXR_ViewType_None，textureQueue 为销毁的纹理队列句柄，looper 为 GSXR_NULL_HANDLE。

[返回值]

成功

- `GSXR_Result_Success` 纹理队列销毁成功。

失败

- `GSXR_Error_Operation_Failed` 纹理队列销毁失败。
- `GSXR_Error_Handle_Invalid` textureQueue 为无效句柄。
- `GSXR_Error_Call_Flow_Invalid` 调用此函数前未先行调用 GSXR_CreateTextureQueue。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

9.3.2. 纹理的获得与释放

XR Runtime 提供的纹理队列，其中的纹理数目可能各不相同。XR 应用程序可经由调用 GSXR_GetTextureCount 函数获取纹理队列中的纹理数目 count，其后操作纹理队列获取出的纹理元素则描述其所属索引 index，0 <= index <

count。

[函数] GSXR_GetTextureCount (获取纹理队列中的纹理数目)

```
GSXR_Result GSXR_GetTextureCount (
    GSXR_TextureQueue           textureQueue,
    uint32_t*                   count);
```

说明：

获取纹理队列中的纹理数目

参数：

[in] textureQueue GSXR_TextureQueue 句柄，由 GSXR_CreateTextureQueue 获取

[out] count 纹理数目

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetTextureCount

[参数检验]

- `textureQueue` 必须是有效的 GSXR_TextureQueue 句柄。
- `count` 必须是指向 uint32_t 数值的有效指针。

[函数实现]

- 渲染器获取纹理队列 textureQueue 中的纹理数目填入 count 输出。

[返回值]

成功

- `GSXR_Result_Success` 纹理数目获取成功。

失败

- GSXR_Error_Operation_Failed 纹理数目获取失败。
- GSXR_Error_Handle_Invalid textureQueue 为无效句柄。
- GSXR_Error_Argument_Invalid count 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Renderer_Unavailable 渲染器未处于正确工作状态。

纹理队列中的纹理元素数目是有限的，XR 应用程序从纹理队列中获取出纹理进行内容渲染之后，必须将纹理元素释放返还纹理队列，并将此纹理元素提交给渲染器进行 XR 后处理渲染。若 XR 应用程序释放返还纹理的频率小于等待获取纹理的频率，应用程序在获取纹理时便会遭遇等待，无法即时获取可用纹理。

[结构体] GSXR_TextureElement (纹理元素)

```
typedef struct GSXR_TextureElement {
   void*                 next;
   uint32_t              index;
   GSXR_Texture          texture;
} GSXR_TextureElement;
```

说明：

纹理元素结构体，用以描述纹理元素的索引值及纹理句柄

成员：

next 预留指针

index 纹理元素于纹理队列中的索引，从 0 起算

texture 纹理句柄

[函数] GSXR_WaitAndAcquireAvailableTexture (等待获得可用纹理)

```
GSXR_Result GSXR_WaitAndAcquireAvailableTexture (
     GSXR_TextureQueue             textureQueue,
     GSXR_Duration                 timeout,
     GSXR_TextureElement*          textureElement);
```

说明：

等待纹理队列获得可用纹理

参数：

[in] textureQueue GSXR_TextureQueue 句柄，由 GSXR_CreateTextureQueue 获取

[in] timeout 持续等待的超时时限值，单位为 naroseconds

[out] textureElement 指向 GSXR_TextureElement 结构体

返回值：

GSXR_Result_Success

GSXR_Result_Timeout_Expired

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_WaitAndAcquireAvailableTexture

[参数检验]

- `textureQueue` 必须是有效的 GSXR_TextureQueue 句柄。
- `timeout` 必须是有效的 GSXR_Duration 数值。
- `textureElement` 必须是指向 GSXR_TextureElement 数值的有效指针。

[函数实现]

- 渲染器于 timeout 时间内等待从纹理队列 textureQueue 中获得可用的纹理填入 textureElement 输出，timeout 时间内无法获得可用纹理则返回 GSXR_Result_Timeout_Expired。
- 若 timeout 为 GSXR_NO_DURATION，函数被调用时无法即时获得可用纹理需立即返回 GSXR_Result_Timeout_Expired。若 timeout 为 GSXR_INFINITE_DURATION，函数需持续等待至可从纹理队列中获得可用纹理方可返回。

[返回值]

成功

- `GSXR_Result_Success` 可用纹理获得成功。

- `GSXR_Result_Timeout_Expired` 等待 timeout 超时无法获得可用纹理。

失败

- `GSXR_Error_Operation_Failed` 可用纹理获得失败。
- `GSXR_Error_Handle_Invalid` textureQueue 为无效句柄。
- `GSXR_Error_Argument_Invalid` timeout 为无效时长。或 textureElement 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

[函数] GSXR_ReleaseTexture (将纹理释放回纹理队列)

```
GSXR_Result GSXR_ReleaseTexture (
    GSXR_TextureQueue                   textureQueue,
    const GSXR_TextureElement*          textureElement);
```

说明：

将纹理释放回纹理队列

参数：

[in] textureQueue GSXR_TextureQueue 句柄，由 GSXR_CreateTextureQueue 获取

[in] textureElement 指定释放的纹理元素，由

GSXR_WaitAndAcquireAvailableTexture 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_ReleaseTexture

[参数检验]

- `textureQueue` 必须是有效的 GSXR_TextureQueue 句柄。

- `textureElement` 必须是指向 GSXR_TextureElement 数值的有效指针。

[函数实现]

- 渲染器将纹理元素 textureElement 释放回纹理队列 textureQueue 中。

[返回值]

成功

- `GSXR_Result_Success` 纹理元素释放成功。

失败

- `GSXR_Error_Operation_Failed` 纹理元素释放失败。
- `GSXR_Error_Handle_Invalid` textureQueue 为无效句柄。
- `GSXR_Error_Argument_Invalid` textureElement 为 nullptr 或检验为无效的纹理元素。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

9.4. 渲染循环

在成功创建渲染器，获取视图集配置，并创建纹理队列之后，XR 应用程序便可开始进行渲染循环，将渲染帧显示给用户。

本章首节说明一个标准的渲染循环遍历的各种状态，以及进入及离开各状态的必要条件，而后的各节则依序介绍构建渲染循环所必需使用的相关函数，包含如何开始与停止一个渲染循环，如何操作帧同步并获取帧显示时间点的视图姿态进行渲染，以及渲染合成层的建构与渲染帧的提交，最后介绍如何使用延伸渲染功能函数。

9.4.1. 渲染循环状态

渲染循环状态各自代表着不同的运行情况，各状态也有各自的形成条件，XR Runtime 必须根据当前状态发送状态事件给应用程序，应用程序必须根据

Runtime 上报的状态事件采取适当的处置，才可确保渲染循环正确运行。

此节介绍八种渲染循环状态的定义，以及 Runtime 及应用程序在不同状态所应采取的处理动作，后续章节将逐步介绍构建整个渲染循环所运用到的所有相关函数。

1. Available 状态

当 XR 应用程序成功调用 GSXR_CreateRenderer 创建渲染器，且系统及显示设备所对应的视图集状态正常，渲染循环即进入 Available 状态，代表此视图及其渲染循环已经处于可用状态。此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Available 事件给应用程序，当应用程序获取此事件时即可调用 GSXR_StartRenderLooper 开始渲染循环。

2. Starting 状态

当 XR 应用程序成功调用 GSXR_StartRenderLooper 开始渲染循环，渲染循环即进入 Starting 状态，代表此渲染循环处于起始中状态。此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Starting 事件给应用程序，并根据 Runtime 自身的设计检视进入下一 Ready 状态的必要条件，如 TextureQueue 是否已创建、空间原点是否已设置等。

3. Ready 状态

当 XR Runtime 备妥渲染循环的必要资源，也确认 XR 应用程序完成所有必要动作，渲染循环即进入 Ready 状态，代表此渲染循环处于已备妥状态。此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Ready 事件给应用程序，应用程序便可开始进行应用的帧循环渲染。

4. Focused 状态

当 XR 应用程序完成了第一次的帧渲染，并将渲染帧提交给 XR Runtime 之后，渲染循环即进入 Focused 状态，代表应用内容已经可见且当前输入焦点属于应用程序（在此状态，应用程序可以接收到设备输入数据进行内容交互），此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Focused 事件给应用程序。

5. Unfocused 状态

当 XR Runtime 将系统渲染层（例如：系统选单）覆盖于应用程序合成层的上并从应用程序身上获取输入焦点后，渲染循环即进入 Unfocused 状态，代表应用内容虽可见但当前输入焦点不属于应用程序（在此状态，应用程序无法接收到设备输入数据进行内容交互），此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Unfocused 事件给应用程序。

6. Unready 状态

当 XR Runtime 检测到系统或设备发生异常或断线时，渲染循环即进入 Unready 状态，代表渲染循环已无法正常工作，处于未备妥状态，此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Unready 事件给应用程序，当应用程序获取此事件时必须调用 GSXR_StopRenderLooper 停止渲染循环。

7. Stopping 状态

当 XR 应用程序成功调用 GSXR_StopRenderLooper 结束渲染循环，渲染循环即进入 Stopping 状态，代表此渲染循环处于结束中状态。此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Stopping 事件给应用程序，并释放此渲染循环的系统资源。

8. Unavailable 状态

当 XR Runtime 释放完渲染循环的必要资源，渲染循环即进入 Unavailable 状态，代表此渲染循环已经处于不可用状态，此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Unavailable 事件给应用程序。待 XR Runtime 确认系统及设备状态正常后再重新进入 Available 状态。

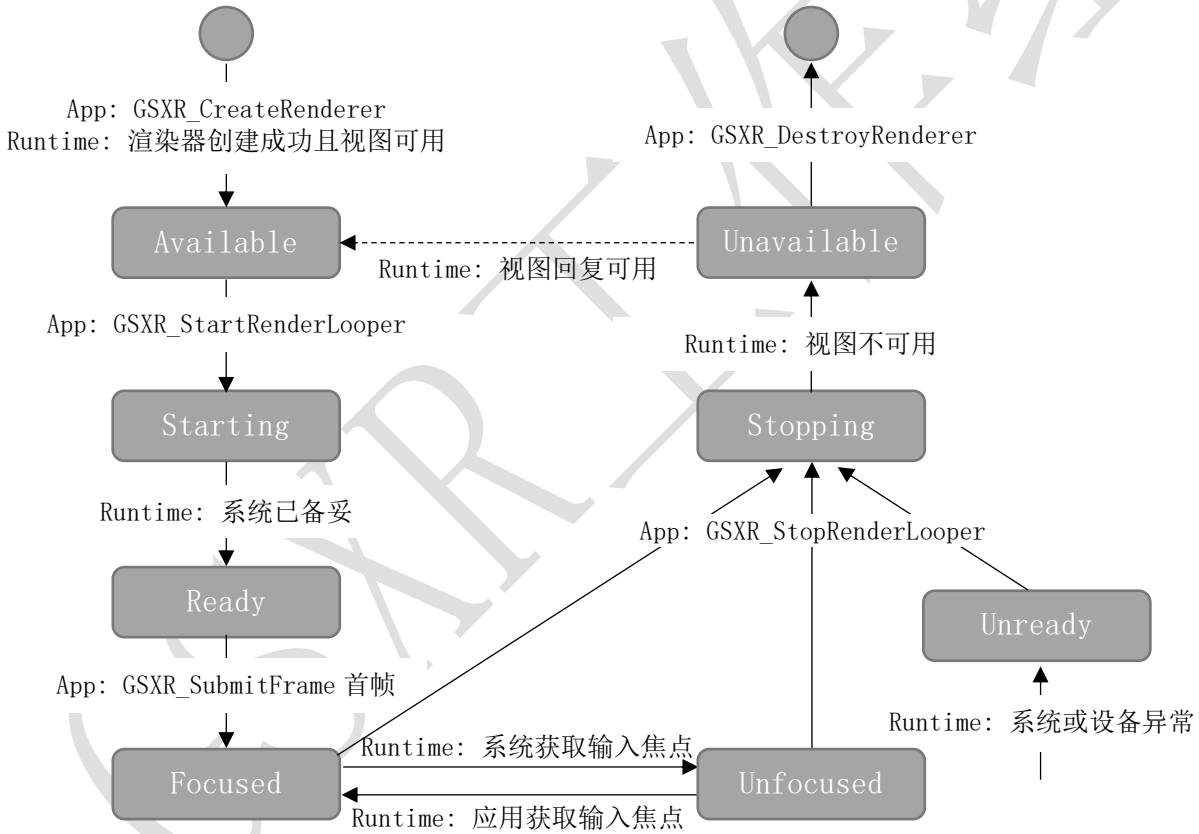
[基础能力规范] 渲染循环状态事件

[功能实现]

- GSXR_EventType_RendererState_Looper_Available 当渲染循环进入 Available 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Available 事件到事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄 GSXR_Renderer, viewType 为状态可用的视图类型 GSXR_ViewType，textureQueue 及 looper 为 GSXR_NULL_HANDLE。
- GSXR_EventType_RendererState_Looper_Starting 当渲染循环进入 Starting 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Starting 事件到事件队列，GSXR_RendererStateEvent 的成员除 textureQueue 为 GSXR_NULL_HANDLE 外全部须有值。
- GSXR_EventType_RendererState_Looper_Ready 当渲染循环进入 Ready 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Ready 事件到事件队列，GSXR_RendererStateEvent 的成员除 textureQueue 为 GSXR_NULL_HANDLE 外全部须有值。
- GSXR_EventType_RendererState_Looper_Focused 当渲染循环进入 Focused 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Focused 事件到事件队列，GSXR_RendererStateEvent 的成员除 textureQueue 为 GSXR_NULL_HANDLE 外全部须有值。
- GSXR_EventType_RendererState_Looper_Unfocused 当渲染循环进入 Unfocused 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Unfocused 事件到事件队列，GSXR_RendererStateEvent 的成员除 textureQueue 为 GSXR_NULL_HANDLE 外全部须有值。
- GSXR_EventType_RendererState_Looper_Unready 当渲染循环进入 Unready 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Unready 事件到事件队列，GSXR_RendererStateEvent 的成员除 textureQueue 为 GSXR_NULL_HANDLE 外全部须有值。
- GSXR_EventType_RendererState_Looper_Sopping 当渲染循环进入 Stopping 状态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Stopping 事件到事件队列，GSXR_RendererStateEvent 的成员除 textureQueue 为 GSXR_NULL_HANDLE 外全部须有值。
- GSXR_EventType_RendererState_Looper_Unavailable 当渲染循环进入 Unavailable 状

态，XR Runtime 发送 GSXR_EventType_RendererState_Looper_Unavailable 事件到事件队列，GSXR_RendererStateEvent 的成员 renderer 为渲染器句柄 GSXR_Renderer，viewType 为状态不可用的视图类型 GSXR_ViewType，textureQueue 为 GSXR_NULL_HANDLE，looper 为停用的渲染循环句柄 GSXR_RenderLooper。

渲染循环的生命周期：



9.4.2. 渲染循环的开始与停止

XR 应用程序必须指定一个视图类型以及该视图类型所支持的功能选项做为初始渲染循环的参数，XR Runtime 必须根据应用的输入参数初始渲染循环，并

输出其渲染循环句柄供 XR 应用后续调用渲染循环函数使用。当 XR 应用离开前或遭遇系统或设备异常时，XR 应用程序必须停止渲染循环，将资源返还系统。

渲染循环功能选项为 XR Runtime 提供给 XR 应用程序选用的渲染循环功能，这些功能将决定 XR Runtime 在执行 XR 渲染后处理时的渲染管线运行流程。XR 应用程序可经由 GSXR_GetRenderLooperOptions 函数获取 XR Runtime 所支持的渲染循环功能选项，若无特殊需求，应用程序应直接使用系统建议的渲染循环功能选项开始渲染循环。

[枚举] GSXR_ColorSpace (颜色空间)

名称	数值	描述
GSXR_ColorSpace_sRGB	0x00000001	sRGB
GSXR_ColorSpqce_Linear	0x00000002	Linear

[结构体] GSXR_AndrosOpenGLESBinding (OpenGL ES 特化信息)

```
#define GSXR_EGL_EXTENSION_MAX_SIZE 16
typedef int64_t GSXR_EglExtension;
typedef struct GSXR_AndrosOpenGLESBinding {
   void*                 next;
   void*                 eglDisplay;
   void*                 eglContext;
   uint32_t              eglExtensionsCount;
   GSXR_EglExtension     eglExtensions[GSXR_EGL_EXTENSION_MAX_SIZE];
} GSXR_AndrosOpenGLESBinding;
```

说明：

OpenGL ES 特化信息结构体，用以提供 Runtime 需求的 OpenGL ES 信息

成员：

next 预留指针

eglDisplay OpenGL ES EGLDisplay

eglContext OpenGL ES EGLContext

eglExtensionsCount 有效的 EGL Extension 数组个数，通常为 0，最大为 16

eglExtensions 设置应用需求的 EGL Extension 定义数值

[结构体] GSXR_RenderLooperStartInfo (渲染循环开始资讯)

```
typedef struct GSXR_RenderLooperStartInfo {
   void*                 next;
   GSXR_ViewType         viewType;
   const void*           graphicsBinding;
   void*                 m_androsAppNativeWindow;
   GSXR_ColorSpace       m_colorSpace;
} GSXR_RenderLooperStartInfo;
```

说明：

渲染循环开始资讯结构体，用以描述渲染循环起始参数

成员：

next 预留指针

viewType 设置渲染循环所对应的视图类型

连接图形 API 特化资讯（在移动设备系统，graphicsBinding 指向 GSXR_AndrosOpenGLESBinding 结构体）

m_androsAppNativeWindow 应用的 ANativeWindow

m_colorSpace 应用的颜色空间配置

渲染循环在起始完成后以句柄识别，句柄定义如下：

```
GSXR_DEFINE_HANDLE(GSXR_RenderLooper)
```

[函数] GSXR_StartRenderLooper (开始渲染循环)

```
GSXR_Result GSXR_StartRenderLooper (
    GSXR_Renderer                          renderer,
```

```
    const GSXR_RenderLooperStartInfo*       startInfo,
    GSXR_RenderLooper*                      looper);
```

说明：

开始渲染循环

参数：

[in] renderer GSXR_Renderer 句柄，由 GSXR_CreateRenderer 获取

[in] startInfo 指向 GSXR_RenderLooperStartInfo 结构体，内含渲染循环开始资讯

[out] looper 渲染循环创建成功输出一个有效句柄，创建失败输出 GSXR_NULL_HANDLE

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

渲染器状态需处于 GSXR_EventType_RendererState_Available 方能开始渲染循环

[基础能力规范] GSXR_StartRenderLooper

[参数检验]

- `renderer` 必须是有效的 GSXR_Renderer 句柄。
- `startInfo` 必须是指向 GSXR_RenderLooperStartInfo 结构的有效指针。
- `looper` 必须是指向 GSXR_RenderLooper 数值的有效指针。

[函数实现]

- 渲染器 renderer 根据 startInfo 资讯开始渲染循环，并将句柄填入 looper 输出。

[返回值]

成功

- `GSXR_Result_Success` 渲染循环起始成功。

失败

- `GSXR_Error_Operation_Failed` 渲染循环起始失败。
- `GSXR_Error_Handle_Invalid` renderer 为无效句柄。
- `GSXR_Error_Argument_Invalid` startInfo 为 nullptr 或检验为无效的输入参数。或 looper 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

[函数] GSXR_StopRenderLooper (停止渲染循环)

```
GSXR_Result GSXR_StopRenderLooper (
    GSXR_RenderLooper       looper);
```

说明：

停止渲染循环

参数：

[in] looper GSXR_RenderLooper 句柄, 由 GSXR_StartRenderLooper 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_StopRenderLooper

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。

[函数实现]

- 渲染器停止渲染循环 looper。

[返回值]

成功

- GSXR_Result_Success 渲染循环停止成功。

失败

- GSXR_Error_Operation_Failed 渲染循环停止失败。
- GSXR_Error_Handle_Invalid looper 为无效句柄。
- GSXR_Error_Call_Flow_Invalid 调用此函数前未先行调用 GSXR_StartRenderLooper。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Renderer_Unavailable 渲染器未处于正确工作状态。

[函数] GSXR_SetRenderLooperThread (设置渲染循环线程 id)

```
GSXR_Result GSXR_SetRenderLooperThread (
    GSXR_RenderLooper      looper,
    uint32_t               threadId);
```

说明：

设置 XR 应用渲染循环线程 id, Runtime 赋予此线程较高的线程优先权

参数：

[in] looper GSXR_RenderLooper 句柄, 由 GSXR_StartRenderLooper 获取

[in] threadId 应用渲染循环线程 id

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_SetRenderLooperThread

[参数检验]

- looper 必须是有效的 GSXR_RenderLooper 句柄。
- threadId 必须是有效的线程 id。

[函数实现]

- Runtime 赋予 threadId 较高的线程优先权。
- 一个 looper 只允许指定一个 threadId，后指定的 threadId 将覆盖前指定。

[返回值]

成功

- GSXR_Result_Success 渲染线程设置成功。

失败

- GSXR_Error_Operation_Failed 渲染线程设置失败。
- GSXR_Error_Handle_Invalid looper 为无效句柄。
- GSXR_Error_Argument_Invalid threadId 为无效线程。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Renderer_Unavailable 渲染器未处于正确工作状态。

9.4.3. 帧同步与视图姿态

为使 XR 应用程序的渲染帧确实与显示同步，GSXR 提供了 GSXR_WaitFrame 函数限制渲染循环，XR Runtime 根据实际显示设备的刷新节奏调节应用程序调用 GSXR_WaitFrame 后的返回时机，并在返回函数时输出下一渲染帧的预测显示时间。XR 应用程序必须以此预测时间调用 GSXR_GetPoseState 获取设备姿态，以及调用 GSXR_GetViewPoseState 获取视图姿态，以预测该帧显示时的姿态进行内容渲染，补偿渲染与显示之间的时间差所造成的延迟。

[函数] GSXR_WaitFrame (等待调节帧渲染时机)

```
GSXR_Result GSXR_WaitFrame (
    GSXR_RenderLooper       looper,
    GSXR_Time*              predictedDisplayTime);
```

说明：

等待调节帧渲染时机，并输出渲染帧的预测显示时间， XR 应用需依据此时间获取对应姿态进行渲染

参数：

[in] looper GSXR_RenderLooper 句柄，由 GSXR_StartRenderLooper 获取

[out] predictedDisplayTime 输出应用渲染此帧后的显示时间

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_WaitFrame

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。
- `predictedDisplayTime` 必须是指向 GSXR_Time 数值的有效指针。

[函数实现]

- XR Runtime 根据显示频率调节帧渲染时机返回函数，并于 predictedDisplayTime 输出渲染帧的预测显示时间。

[返回值]

成功

- `GSXR_Result_Success` 调节成功。

失败

- `GSXR_Error_Operation_Failed` 调节失败。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` predictedDisplayTime 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

XR 应用程序要正确渲染出适用 XR 显示设备的视图，必须先获取视图姿态以

及投影参数。在开始渲染循环时所输入的视图形态（GSXR_ViewType）所对应的视图集配置（GSXR_ViewSetConfiguration），其成员视图数目（viewCount）便是 XR 应用程序调用 GSXR_GetViewPoseState 函数获取视图姿态的个数，XR 应用程序需根据各个视图的视图姿态及投影参数进行内容渲染，再将渲染帧提交给 Runtime。

[结构体] GSXR_Fovf（视场角）

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

说明：

视场角结构体，用以描述视图位姿可见的视图区域范围

成员：

angleLeft 左视场角，单位为弧度

angleRight 右视场角，单位为弧度

angleUp 上视场角，单位为弧度

angleDown 下视场角，单位为弧度

[结构体] GSXR_ViewPoseState（视图姿态）

```
typedef struct GSXR_ViewPoseState {
   GSXR_Posef              pose;
   GSXR_Fovf               fov;
} GSXR_ViewPoseState;
```

说明：

视图姿态结构体，用以描述视图位姿及其可见视场角

成员：

pose 位姿结构体

fov 视场角结构体

[函数] GSXR_GetViewPoseState (获取视图于指定时间的姿态)

```
GSXR_Result GSXR_GetViewPoseState (
    GSXR_RenderLooper           looper,
    GSXR_Time                   specifiedTime,
    uint32_t                    viewCount,
    GSXR_ViewPoseState*         viewPoseState);
```

说明：

获取视图于指定时间的姿态

参数：

[in] looper GSXR_RenderLooper 句柄，由 GSXR_StartRenderLooper 获取

[in] specifiedTime 指定时间

[in] viewCount viewPoseState 数组个数，需相等于 GSXR_ViewSetConfiguration 的 viewCount 数

[out] viewPoseState 指向 GSXR_ViewPoseState 结构体的数组

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetViewPoseState

[参数检验]

- looper 必须是有效的 GSXR_RenderLooper 句柄。
- specifiedTime 必须是有效的时间戳数值。
- viewCount 必须相等于 GSXR_ViewSetConfiguration 的 viewCount 数。
- viewPoseState 必须是指向 GSXR_ViewPoseState 结构的有效指针。

[函数实现]

- Runtime 根据 specifiedTime 获取 XR 显示设备的姿态，再依据其对应视图转换成视图姿态填入 viewPoseState 输出，视图姿态数组个数 viewCount 需相等于视图集配置的视图数目。

[返回值]

成功

- `GSXR_Result_Success` 视图姿态获取成功。

失败

- `GSXR_Error_Operation_Failed` 视图姿态获取失败。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` specifiedTime 为无效时间戳。或 viewCount 不等于 GSXR_ViewSetConfiguration 的 viewCount 数。或 viewPoseState 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

9.4.4. 帧提交与合成层

XR 应用程序在等待 GSXR_WaitFrame 返回并完成内容渲染后，必须调用渲染帧提交函数 GSXR_SubmitFrame 将渲染帧提交给 Runtime 渲染管线进行 XR 渲染后处理。

[结构体] GSXR_FrameSubmitInfo（渲染帧提交信息）

```
typedef struct GSXR_FrameSubmitInfo {
    void*                                       next;
    GSXR_Time                                   displayTime;
    uint32_t                                    layerCount;
    const GSXR_CompositionLayerHeader* const*   layers;
} GSXR_FrameSubmitInfo;
```

说明：

渲染帧提交信息结构体，用以描述渲染帧的合成层信息

成员：

next 预留指针

displayTime 渲染帧的显示时间

layerCount 合成层数

layers 合成层指针的数组

[函数] GSXR_SubmitFrame (提交渲染帧)

```
GSXR_Result GSXR_SubmitFrame (
    GSXR_RenderLooper                   looper,
    const GSXR_FrameSubmitInfo*         submitInfo);
```

说明：

提交应用的渲染帧信息给 Runtime 渲染管线进行后续处理

参数：

[in] looper GSXR_RenderLooper 句柄, 由 GSXR_StartRenderLooper 获取

[in] submitInfo 指向 GSXR_FrameSubmitInfo 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_SubmitFrame

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。
- `submitInfo` 必须是指向 GSXR_FrameSubmitInfo 结构的有效指针。

[函数实现]

- Runtime 根据 submitInfo 的合成层信息进行合成并进行 XR 渲染后处理，并输出最后合成帧给显示设备。
- 若 submitInfo 的 layerCount 为 0，Runtime 清除当前显示内容。

[返回值]

成功

- `GSXR_Result_Success` 帧提交成功。

失败

- `GSXR_Error_Operation_Failed` 帧提交失败。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` submitInfo 为 nullptr 或检验为无效的输入参数。
- `GSXR_Error_Call_Flow_Invalid` 调用此函数前未先行调用 GSXR_WaitFrame。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

XR 应用程序提交的渲染帧信息，可包含一个或多个合成层，再由 XR Runtime 进行合成，若输入的 GSXR_FrameSubmitInfo 的 layerCount 为 0，则 XR Runtime 必须清除当前的显示内容。

XR 应用程序必须根据合成层类型以及内容需求，选用对应的合成层结构体，备齐该合成层结构体所需的功能选项及参数，并指定此合成层所对应的渲染纹理子图像及视图姿态，将完整的合成层信息正确记录于 submitInfo，方可调用 GSXR_SubmitFrame 函数提交渲染帧。

合成层结构体全部都把 GSXR_CompositionLayerHeader 结构作为结构标头，指定其合成层类型及合成层功能选项，合成层结构标头后的其余成员则根据该合成层类型分别制定，XR 应用程序必须确实输入正确的合成层信息，XR Runtime 方可正确输出最后的合成结果。

[枚举] GSXR_CompositionLayerType (合成层类型)

名称	数值	描述
GSXR_CompositionLayerType_Projection	0	三维场景投影层
GSXR_CompositionLayerType_Quad	1	二维四方平面层

[枚举] GSXR_CompositionLayerOptions (合成层功能选项)

名称	数值	描述
GSXR_CompositionLayerOptions_Distortion_Correction	0x00000001	畸变校正
GSXR_CompositionLayerOptions_Alpha_Blending	0x00000002	Alpha 混合

[结构体] GSXR_CompositionLayerHeader (渲染合成层标头)

```
typedef struct GSXR_CompositionLayerHeader {
    void*                           next;
    GSXR_CompositionLayerType       layerType;
    GSXR_Flags64                    layerOptions;
} GSXR_CompositionLayerHeader;
```

说明:

渲染合成层标头结构体, 用以描述合成层类型及功能选项

成员:

next 预留指针

layerType 合成层类型

layerOptions 设置启用的合成层功能选项

[结构体] GSXR_TexSubImage (纹理子图像)

```
typedef struct GSXR_TexSubImage {
    void*                       next;
    GSXR_TextureQueue           textureQueue;
    GSXR_TextureElement         textureElement;
    int32_t                     xOffset;
```

```
    int32_t                     yOffset;
    int32_t                     width;
    int32_t                     height;
    uint32_t                    arrayLayerIndex;
} GSXR_TexSubImage;
```

说明：

纹理子图像结构体，用以描述合成层所对应的纹理信息

成员：

next 预留指针

textureQueue 纹理队列句柄

textureElement 纹理元素

xOffset 纹理子图像 x 偏移量

yOffset 纹理子图像 y 偏移量

width 纹理子图像宽度

height 纹理子图像高度

arrayLayerIndex 纹理子图像数组索引

[枚举]GSXR_Eye

```
typedef enum GSXR_Eye {
    GSXR_Eye_Left                   1
    GSXR_Eye_Right                  2
    GSXR_Eye_Both                   3
} GSXR_Eye;
```

说明：

合成层视图信息结构体，用以描述合成层所对应的视图信息

成员：

GSXR_Eye_Left 左眼

GSXR_Eye_Right 右眼

GSXR_Eye_Both 双眼

[结构体] GSXR_CompositionLayerView（合成层视图信息）

```
typedef struct GSXR_CompositionLayerView {
```

```
    void*                                   next;
    GSXR_ViewPoseState                      viewPoseState;
    GSXR_TexSubImage                        texSubImage;
    GSXR_Eye                                eyeMask
} GSXR_CompositionLayerView;
```

说明：

合成层视图信息结构体，用以描述合成层所对应的视图信息

成员：

next 预留指针

viewPoseState 视图姿态

texSubImage 纹理子图像信息

eyeMask 左右眼标识

基础合成层类型有二：

1. 三维场景投影层

从人眼视点使用透视投影所构成的平面投影图像，通常用于渲染虚拟场景。

2. 二维四方平面层

在虚拟场景中显示的二维矩形平面，通常用于渲染选单或仪表板类的二维平面物件。由于二维平面通常只占据视野的一小部份，应用程序可以透过指定较高分辨率的纹理尺寸，赋予该平面更好的可读性。且将二维平面层抽离三维场景直接由 Runtime 进行合成，也可以减少多次采样所造成的失真效果。此外，二维平面允许指定其平面位姿所相对的空间原点位置是否以视图位姿为原点，以创建跟随头部运动的二维平面物件。

[结构体] GSXR_CompositionLayerProjection（三维场景投影层）

```
typedef struct GSXR_CompositionLayerProjection {
    void*                                   next;
    GSXR_CompositionLayerType               layerType;
```

```
    GSXR_Flags64                                layerOptions;
    uint32_t                                    viewCount;
    const GSXR_CompositionLayerView*            views;
} GSXR_CompositionLayerProjection;
```

说明：

三维场景投影层结构体，用以描述对三维场景使用透视投影所构成的平面图像

成员：

next 预留指针

layerType 合成层类型，GSXR_CompositionLayerType_Projection

layerOptions 设置启用的合成层功能选项

viewCount 视图数目

views 投影层视图内容数组

[结构体] GSXR_CompositionLayerQuad（二维四方平面层）

```
typedef struct GSXR_CompositionLayerQuad {
    void*                                       next;
    GSXR_CompositionLayerType                   layerType;
    GSXR_Flags64                                layerOptions;
    uint32_t                                    viewCount;
    const GSXR_CompositionLayerView*            views;
    GSXR_Bool32                                 isOriginOnView;
    GSXR_Posef                                  quadPose;
    float                                       width;
    float                                       height;
} GSXR_CompositionLayerQuad;
```

说明：

二维四方平面层结构体，用以描述二维矩形平面的合成层信息

成员：

next 预留指针

layerType 合成层类型，GSXR_CompositionLayerType_Quad

layerOptions 设置启用的合成层功能选项

viewCount 视图数目

views 投影层视图内容数组

isOriginOnView 平面层位姿是否以视图为原点

quadPose 平面层位姿

width 平面层宽度，单位为米

height 平面层高度，单位为米

9.4.5. 预设置与后设置渲染功能

除上述介绍的基础渲染循环函数功能之外，GSXR 也会提供延伸的渲染功能函数，供 XR 应用程序于内容渲染时启用 XR Runtime 支持的延伸渲染功能。渲染功能类型定义于 GSXR_RenderFunctionType 中。

[枚举] GSXR_RenderFunctionType (渲染功能类型)

名称	数值	描述
GSXR_RenderFunctionType_Foveated	0x00000001	注视点渲染

[函数] GSXR_GetRenderFunctions (获取渲染功能)

```
GSXR_Result GSXR_GetRenderFunctions (
    GSXR_RenderLooper          looper,
    GSXR_Flags64*              recommendedFunctions,
    GSXR_Flags64*              supportedFunctions);
```

说明：

获取渲染功能，包含建议的及所有的支持功能

参数：

[in] looper GSXR_RenderLooper 句柄，由 GSXR_StartRenderLooper 获取

[out] recommendedFunctions GSXR_RenderFunctionType 的位掩码，输出系统建议的渲染功能

[out] supportedFunctions GSXR_RenderFunctionType 的位掩码，输出可支持的所有渲染功能

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetRenderFunctions

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。
- `recommendedFunctions` 必须是指向 GSXR_Flags64 数值的有效指针。
- `supportedFunctions` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- 获取渲染循环 looper 所支持的渲染功能，将建议的以及所有的支持功能各自进行 OR 运算后填入 recommendedFunctions 及 supportedFunctions 输出。

[返回值]

成功

- `GSXR_Result_Success` 渲染功能获取成功。

失败

- `GSXR_Error_Operation_Failed` 渲染功能获取失败。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` recommendedFunctions 为 nullptr。或 supportedFunctions 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

XR 应用程序使用的延伸渲染功能信息，需于 GSXR_FrameRenderInfo 中指定使用功能的所需参数，并在 GSXR_WaitFrame 之后及应用程序开始渲染帧内容之前，调用 GSXR_PreRenderFrame 预设置此内容帧所要使用延伸渲染功能，下一帧

循环也会持续延用此设定，直至应用程序于 GSXR_PostRenderFrame 后设置停用此功能。

GSXR_PreRenderFrame 及 GSXR_PostRenderFrame 并不是 XR 渲染循环的必须函数，XR 应用程序可根据自身内容的需求选择使用与否，使用时 GSXR_FrameRenderInfo 的 functionCount 不可为 0，且指定使用的延伸渲染功能必须是 XR Runtime 所支持的功能。

[结构体] GSXR_FrameRenderInfo（渲染帧功能信息）

```
typedef struct GSXR_FrameRenderInfo {
   void*                                    next;
   uint32_t                                 functionCount;
   const GSXR_RenderFunctionHeader* const*  functions;
} GSXR_FrameRenderInfo;
```

说明：

渲染帧功能信息结构体，用以描述应用程序渲染帧使用的延伸渲染功能信息

成员：

next 预留指针

functionCount 渲染功能数目

functions 渲染功能指针的数组

[函数] GSXR_PreRenderFrame（预设置渲染帧功能）

```
GSXR_Result GSXR_PreRenderFrame (
     GSXR_RenderLooper                  looper,
     const GSXR_FrameRenderInfo*        preRenderInfo);
```

说明：

于应用进行渲染前，预设置渲染功能参数予应用的渲染纹理，通常用于渲染循环中启用渲染功能

参数：

[in] looper GSXR_RenderLooper 句柄，由 GSXR_StartRenderLooper 获取

[in] preRenderInfo 指向 GSXR_FrameRenderInfo 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

必须在应用进行渲染前调用

[基础能力规范] GSXR_PreRenderFrame

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。
- `preRenderInfo` 必须是指向 GSXR_FrameRenderInfo 结构的有效指针。

[函数实现]

- Runtime 根据 preRenderInfo 的预设置功能信息，对其指定的纹理设置此延伸渲染功能。
- preRenderInfo 的 functionCount 不可为 0。指定的功能必须是 Runtime 所支持。

[返回值]

成功

- `GSXR_Result_Success` 预设置成功。

失败

- `GSXR_Error_Operation_Failed` 预设置失败。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` preRenderInfo 为 nullptr 或检验为无效的输入参数。
- `GSXR_Error_Call_Flow_Invalid` 调用此函数前未先行调用 GSXR_WaitFrame。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

[函数] GSXR_PostRenderFrame（后设置渲染帧功能）

```
GSXR_Result GSXR_PostRenderFrame (
    GSXR_RenderLooper                   looper,
    const GSXR_FrameRenderInfo*         postRenderInfo);
```

说明：

当应用进行渲染后，设置渲染功能参数在应用的渲染纹理上，通常用于渲染循环中停用渲染功能

参数：

[in] looper GSXR_RenderLooper 句柄，由 GSXR_StartRenderLooper 获取

[in] postRenderInfo 指向 GSXR_FrameRenderInfo 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

必须在应用进行渲染后调用

[基础能力规范] GSXR_PostRenderFrame

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。
- `postRenderInfo` 必须是指向 GSXR_FrameRenderInfo 结构的有效指针。

[函数实现]

- Runtime 根据 postRenderInfo 之后设置功能信息，对其指定的纹理设置此延伸渲染功能。
- postRenderInfo 的 functionCount 不可为 0。指定的功能必须是 Runtime 所支持。

[返回值]

成功

- `GSXR_Result_Success` 后设置成功。

失败

- `GSXR_Error_Operation_Failed` 后设置失败。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` postRenderInfo 为 nullptr 或检验为无效的输入参数。
- `GSXR_Error_Call_Flow_Invalid` 调用此函数前未先行调用 GSXR_WaitFrame。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

渲染帧功能结构体全部都把 GSXR_RenderFunctionHeader 结构作为结构标头，指定其渲染功能类型及启用或停用，渲染帧功能结构标头后的其余成员则根据该渲染功能类型分别制定，XR 应用程序必须确实输入正确的延伸渲染功能信息，XR Runtime 才可正确设置予指定的纹理。

[结构体] GSXR_RenderFunctionHeader (渲染功能标头)

```
typedef struct GSXR_RenderFunctionHeader {
   void*                                    next;
   GSXR_RenderFunctionType                  functionType;
   GSXR_Bool32                              enable;
} GSXR_RenderFunctionHeader;
```

说明：

渲染功能标头结构体，用以描述延伸渲染功能类型及是否启用

成员：

next 预留指针

functionType 渲染功能类型

enable GSXR_TRUE 表示启用此渲染功能(通常用于 GSXR_PreRenderFrame)，

GSXR_FALSE 表示停用此渲染功能(通常用于 GSXR_PostRenderFrame)

9.4.6. 注视点渲染功能

GPU 及其图形 API 若支持注视点渲染功能，则 XR 应用程序便可使用此功能针对非注视点区域进行较低解析度的内容渲染，以节省渲染耗费的资源及时间。

注视点参数信息通过 GSXR_FoveatedParameters 加以描述，其 (focalX, focalY) 坐标以眼球中心为平面原点，并将两侧区间归一化成-1 到 1 之间的数值，注视点 (focalX, focalY) 坐标便落于此区间之中，再加上此注视点视场角的角度信息 foveaFov，及非注视点区域的视觉品质 peripheralQuality，即形成完整的注视点参数信息。

[枚举] GSXR_PeripheralQuality (注视点周围视觉品质等级)

名称	数值	描述
GSXR_PeripheralQuality_System	0	系统预设视觉品质等级
GSXR_PeripheralQuality_High	1	高视觉品质等级 (注视点周围轻度模糊)
GSXR_PeripheralQuality_Medium	2	中视觉品质等级 (注视点周围中度模糊)
GSXR_PeripheralQuality_Low	3	低视觉品质等级 (注视点周围重度模糊)

[结构体] GSXR_FoveatedParameters (注视点渲染参数)

```
typedef struct GSXR_FoveatedParameters {
    void*                       next;
    float                       focalX;
    float                       focalY;
    GSXR_Fovf                   foveaFov;
    GSXR_PeripheralQuality      peripheralQuality;
} GSXR_FoveatedParameters;
```

说明：

注视点渲染参数结构体，用以描述注视点的位置、角度、及周围视觉品质等级

成员：

next 预留指针

focalX 注视点 X 坐标，数据范围为 -1 <= focalX <= 1

focalY 注视点 Y 坐标，数据范围为 -1 <= focalY <= 1

foveaFov 注视点视场角

peripheralQuality 注视点周围视觉品质等级

以此注视点参数，再指定此注视点参数所要套用的纹理子图像，形成注视点视图信息结构 GSXR_FoveatedView，Runtime 便可根据此结构获取各视图注视点参数，并将其对应纹理设置成启用注视点渲染。

[结构体] GSXR_FoveatedView（注视点视图信息）

```
typedef struct GSXR_FoveatedView {
   void*                                 next;
   GSXR_TexSubImage                      texSubImage;
   const GSXR_FoveatedParameters*        foveatedParams;
} GSXR_FoveatedView;
```

说明：

注视点视图信息结构体，用以描述注视点渲染参数及其纹理子图像信息

成员：

next 预留指针

texSubImage 纹理子图像信息

foveatedParams 注视点渲染参数

[结构体] GSXR_RenderFunctionFoveated（注视点渲染功能）

```
typedef struct GSXR_RenderFunctionFoveated {
   void*                              next;
   GSXR_RenderFunctionType            functionType;
   GSXR_Bool32                        enable;
   uint32_t                           viewCount;
```

```
    const GSXR_FoveatedView*          foveatedViews;
} GSXR_RenderFunctionFoveated;
```

说明：

注视点渲染功能结构体，用以描述各视图的注视点渲染信息以及是否启用注视点渲染

成员：

next 预留指针

functionType 渲染功能类型，GSXR_RenderFunctionType_Foveated

enable GSXR_TRUE 表示启用注视点渲染，GSXR_FALSE 表示停用注视点渲染

viewCount 视图数目

foveatedViews 注视点视图信息

[函数] GSXR_GetFoveatedParameters（获取现存注视点参数）

```
GSXR_Result GSXR_GetFoveatedParameters (
    GSXR_RenderLooper                   looper,
    GSXR_Bool32*                        enabled,
    uint32_t                            viewCount,
    GSXR_FoveatedParameters*            foveatedParams);
```

说明：

获取现存注视点参数

参数：

[in] looper GSXR_RenderLooper 句柄，由 GSXR_StartRenderLooper 获取

[out] enabled 输出注视点渲染是否启用中，GSXR_TRUE 为启用，GSXR_FALSE 为停用

[in] viewCount foveatedParams 数组个数，需相等于 GSXR_ViewSetConfiguration 的 viewCount 数

[out] foveatedParams 指向 GSXR_FoveatedParameters 结构体的数组

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Renderer_Unavailable

备注：

无

[基础能力规范] GSXR_GetFoveatedParameters

[参数检验]

- `looper` 必须是有效的 GSXR_RenderLooper 句柄。
- `enabled` 必须是指向 GSXR_Bool32 数值的有效指针。
- `viewCount` 必须相等于 GSXR_ViewSetConfiguration 的 viewCount 数。
- `foveatedParams` 必须是指向 GSXR_FoveatedParameters 结构的有效指针。

[函数实现]

- Runtime 根据当前注视点参数信息填入 enabled 及 foveatedParams 中输出。

[返回值]

成功

- `GSXR_Result_Success` 注视点参数获取成功。

失败

- `GSXR_Error_Operation_Failed` 注视点参数获取失败。
- `GSXR_Error_Function_Unsupported` 系统不支持注视点渲染功能。
- `GSXR_Error_Handle_Invalid` looper 为无效句柄。
- `GSXR_Error_Argument_Invalid` enabled 为 nullptr。或 foveatedParams 为 nullptr。或 viewCount 不等于 GSXR_ViewSetConfiguration 的 viewCount 数。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Renderer_Unavailable` 渲染器未处于正确工作状态。

10. XR 系统函数说明

XR 应用程序运行时的顺畅度与系统性能有很强的相关性，通常 XR Runtime 运行期间必须根据运行平台的性能等级，在兼顾 XR 应用程序顺畅运行及维持设

备电量之间获取平衡的前提下，调试出适合此 XR 系统的 CPU, GPU 时钟频率。

在交由 XR Runtime 调控系统效能之余，GSXR 也会提供性能函数交由应用程序自行设置 CPU, GPU 性能等级，唯应用程序所设等级所对应的时钟频率仍由 XR Runtime 根据系统性能于设备平台上做出适当调试。

[枚举] GSXR_PerformanceLevel (系统性能等级)

名称	数值	描述
GSXR_PerformanceLevel_System	0	系统预设等级
GSXR_PerformanceLevel_High	1	高性能等级
GSXR_PerformanceLevel_Medium	2	中性能等级
GSXR_PerformanceLevel_Low	3	低性能等级

[函数] GSXR_SetPerformanceLevels (设置 CPU, GPU 性能等级)

```
GSXR_Result GSXR_SetPerformanceLevels (
    GSXR_Runtime                    runtime,
    GSXR_PerformanceLevel           cpuPerfLevel,
    GSXR_PerformanceLevel           gpuPerfLevel);
```

说明：

设置 CPU, GPU 性能等级

参数：

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] cpuPerfLevel 设置 CPU 性能等级

[in] gpuPerfLevel 设置 GPU 性能等级

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_SetPerformanceLevels

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `cpuPerfLevel` 必须是有效的 GSXR_PerformanceLevel 数值。
- `gpuPerfLevel` 必须是有效的 GSXR_PerformanceLevel 数值。

[函数实现]

- Runtime 根据 cpuPerfLevel 及 gpuPerfLevel 等级，调控 CPU，GPU 时钟频率。

[返回值]

成功

- `GSXR_Result_Success` 性能等级设置成功。

失败

- `GSXR_Error_Operation_Failed` 性能等级设置失败。
- `GSXR_Error_Function_Unsupported` 系统不支持应用程序调控 CPU，GPU 性能。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` cpuPerfLevel 未定义于 GSXR_PerformanceLevel 中。或 gpuPerfLevel 未定义于 GSXR_PerformanceLevel 中。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

11. XR 特性函数说明

XR 特性函数是提供 XR 设备函数、XR 渲染器函数之外的延伸特性接口，可以增添应用内容，用来使用更丰富的 XR 功能。XR 特性都依据 XR 设备上的硬件配置和软件算法，但对 XR 特性的支持程度会有不同。本章将于 10.1 节说明 XR 特性的通用函数，并于后续章节逐一介绍各种 XR 特性的函数定义。

11.1. XR 特性通用函数

XR 应用程序在使用 XR 特性之前，需先调用 GSXR_GetSupportedXrFeatures 函数获取当前 XR Runtime 及 XR 设备所支持的 XR 特性类型，应用程序可根据自身内容性质决定使用的 XR 特性，并以 GSXR_IsXrFeatureAvailable 确认将要使用的特性是可用状态。各特性也各自拥有其特性功能选项，应用程序可使用 GSXR_GetXrFeatureOptions 函数分别获取系统建议及所有支持的功能选项，以此为起始特性运行时的功能选项依据。各特性必须经由 GSXR_StartXrFeature 起始运行并获取特性句柄 GSXR_XrFeature，在使用完特性后以 GSXR_StopXrFeature 停止特性运行。

```
GSXR_DEFINE_HANDLE(GSXR_XrFeature)
```

11.1.1. 特性类型

XR 特性类型如下所示，特性类型通常与设备的硬件及算法有关，XR 应用程序只能使用 XR 设备支持的特性类型，故 XR 应用在决定使用的特性之前必须先获取特性是否可被设备支持，才可确保应用程序所需的特性功能正常运行。

[枚举] GSXR_XrFeatureType (XR 特性类型)

名称	数值	描述
GSXR_XrFeatureType_EyeTracking	0x00000001	眼球追踪
GSXR_XrFeatureType_HandTracking	0x00000002	手部追踪
GSXR_XrFeatureType_ControllerModel	0x00000100	手柄模型
GSXR_XrFeatureType_PointCloud	0x00010000	点云数据

GSXR_XrFeatureType_PlaneDetection	0x00020000	平面侦测
GSXR_XrFeatureType_ImageRecognition	0x00040000	图像识别
GSXR_XrFeatureType_FaceRecognition	0x00080000	脸部识别

[函数] GSXR_GetSupportedXrFeatures (获取支持的特性类型)

```
GSXR_Result GSXR_GetSupportedXrFeatures (
    GSXR_Runtime          runtime,
    GSXR_Flags64*         featureTypeFlags);
```

说明：

获取当前支持的所有特性类型

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[out] featureTypeFlags GSXR_XrFeatureType 的位掩码，输出可支持的所有特性类型

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_GetSupportedXrFeatures

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `featureTypeFlags` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 获取当前支持的所有 XR 特性，将其位掩码进行 OR 运算后填入 featureTypeFlags 输出。

[返回值]

成功

- GSXR_Result_Success 特性类型获取成功。

失败

- GSXR_Error_Operation_Failed 特性类型获取失败。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid featureTypeFlags 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

11.1.2. 特性状态

特性状态描述所使用的特性当前的运行情况，XR Runtime 必须在特性进入各种特性状态时，发送特性状态事件给应用程序，应用程序需根据状态事件采取适当处置，正确管理特性的使用时机。

以下为五种 XR 特性状态：

1. Unavailable 状态

当 XR 特性在 XR 设备上的运作条件未备齐，应用程序无法使用特性时，XR 特性处于 Unavailable 状态，代表此特性处于不可用状态。此时 XR 特性需发送 GSXR_EventType_FeatureState_Unavailable 事件给应用程序，提示应用程序此 XR 特性当前无法被使用。若应用程序此时正在使用此特性，必须先行停止此特性，待 XR 特性重新进入 Available 状态后再重新起始使用。

2. Available 状态

当 XR 特性在 XR 设备上的运作条件全部已备齐且检验通过时，XR 特性即进入 Available 状态，代表此特性已经处于可用状态。此时 XR 特性需发送 GSXR_EventType_FeatureState_Available 事件给应用程序，当应用程序

获取此事件时即可调用 GSXR_StartXrFeature 函数起始特性运行。

3. Starting 状态

当 XR 应用程序成功调用 GSXR_StartXrFeature 函数起始特性运行，XR 特性即进入 Starting 状态，代表此特性处于起始中状态。此时 XR 特性需发送 GSXR_EventType_FeatureState_Starting 事件给应用程序，当 XR 特性成功开始即进入下一 Working 状态。

4. Working 状态

当 XR 特性成功起始并开始正常运作，即进入 Wroking 状态。此时 XR 特性需发送 GSXR_EventType_FeatureState_Working 事件给应用程序，应用程序便可开始操作此 XR 特性的功能函数。当 XR 应用程序不再使用特性时，需调用 GSXR_StopXrFeature 停止 XR 特性。

5. Stopping 状态

当 XR 应用程序成功调用 GSXR_StopXrFeature 函数停止 XR 特性，XR 特性即进入 Stopping 状态，代表此特性处于结束中状态。此时 XR 特性需发送 GSXR_EventType_FeatureState_Stopping 事件给应用程序，并开始停止特性运行并释放此特性的系统资源，待 XR 特性正确停止后，XR 特性进入 Unavailable 状态，检验特性条件备齐则重新进入 Available 状态。

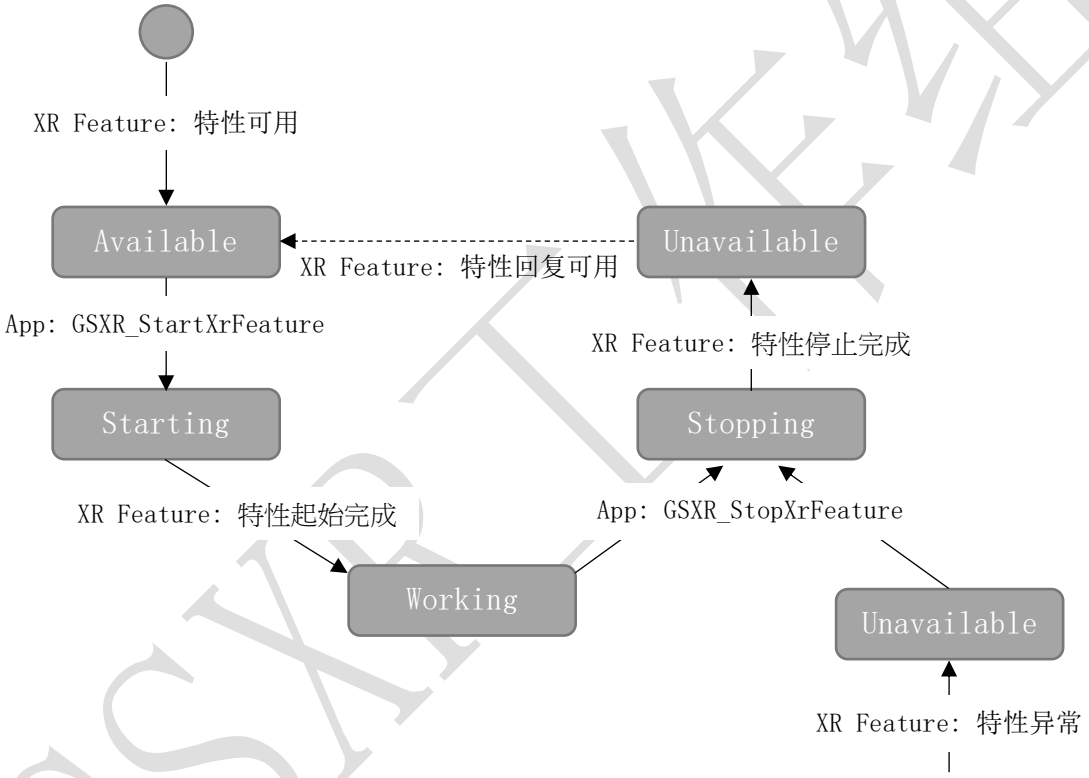
[基础能力规范] XR 特性状态事件

[功能实现]

- GSXR_EventType_FeatureState_Unavailable 当 XR 特性进入 Unavailable 状态，XR 特性发送 GSXR_EventType_FeatureState_Unavailable 事件到事件队列。
- GSXR_EventType_FeatureState_Available 当 XR 特性进入 Available 状态，XR 特性发送 GSXR_EventType_FeatureState_Available 事件到事件队列。

- GSXR_EventType_FeatureState_Starting 当 XR 特性进入 Starting 状态，XR 特性发送 GSXR_EventType_FeatureState_Starting 事件到事件队列。
- GSXR_EventType_FeatureState_Working 当 XR 特性进入 Working 状态，XR 特性发送 GSXR_EventType_FeatureState_Working 事件到事件队列。
- GSXR_EventType_FeatureState_Stopping 当 XR 特性进入 Stopping 状态，XR 特性发送 GSXR_EventType_FeatureState_Stopping 事件到事件队列。

XR 特性的生命周期：



11.1.3. 特性的可用状态

除了从事件队列中获取 XR 特性状态外，XR 应用程序在起始特性运行前，可经由调用 GSXR_IsXrFeatureAvailable 函数获取特性可用状态，若特性处于可用状态才可以开始 XR 特性运行。

[函数] GSXR_IsXrFeatureAvailable (获取特性可用状态)

```
GSXR_Result GSXR_IsXrFeatureAvailable (
    GSXR_Runtime                runtime,
    GSXR_XrFeatureType          featureType,
    GSXR_Bool32*                available);
```

说明：

获取特性可用状态

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] featureType 指定特性类型

[out] available 输出 GSXR_TRUE 表示特性可用，GSXR_FALSE 表示特性不可用

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_IsXrFeatureAvailable

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `featureType` 必须是有效的特性类型。
- `available` 必须是指向 GSXR_Bool32 数值的有效指针。

[函数实现]

- Runtime 获取 featureType 的可用状态填入 available 输出。

[返回值]

成功

- `GSXR_Result_Success` 特性可用状态获取成功。

失败

- `GSXR_Error_Operation_Failed` 特性可用状态获取失败。

- GSXR_Error_Feature_Unsupported 此设备不支持 featureType 特性。
- GSXR_Error_Handle_Invalid runtime 为无效句柄。
- GSXR_Error_Argument_Invalid available 为 nullptr。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。

11.1.4. 特性运行的起始与停止

XR 特性类型所具备的功能选项各不相同，XR 应用程序可以提前获得将要使用的特性功能选项，并根据应用内容本身需求设置特性功能选项起始特性运行，特性功能使用完毕，则应用程序必须停止特性运行，将特性资源返还系统。

[函数] GSXR_GetXrFeatureOptions (获取特性功能选项)

```
GSXR_Result GSXR_GetXrFeatureOptions (
    GSXR_Runtime                runtime,
    GSXR_XrFeatureType          featureType,
    GSXR_Flags64*               recommendedOptions,
    GSXR_Flags64*               supportedOptions);
```

说明：

获取特性的功能选项

参数：

[in] runtime GSXR_Runtime 句柄, 由 GSXR_CreateRuntime 获取

[in] featureType 指定特性类型

[out] recommendedOptions 各特性的功能选项位掩码, 输出系统建议的功能选项

[out] supportedOptions 各特性的功能选项位掩码, 输出可支持的所有功能选项

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

备注：

各特性的功能选项由各特性自行定义

[基础能力规范] GSXR_GetXrFeatureOptions

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `featureType` 必须是有效的特性类型。
- `recommendedOptions` 必须是指向 GSXR_Flags64 数值的有效指针。
- `supportedOptions` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- Runtime 获取 featureType 特性的功能选项，将系统建议的及所有支持的功能选项各自进行 OR 运算后填入 recommendedOptions 及 supportedOptions 输出。

[返回值]

成功

- `GSXR_Result_Success` 特性功能选项获取成功。

失败

- `GSXR_Error_Operation_Failed` 特性功能选项获取失败。
- `GSXR_Error_Feature_Unsupported` 此设备不支持 featureType 特性。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` recommendedOptions 或 supportedOptions 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[结构体] GSXR_XrFeatureStartInfo (特性初始信息)

```
typedef struct GSXR_XrFeatureStartInfo {
   void*                       next;
   GSXR_XrFeatureType          featureType;
   GSXR_Flags64                options;
} GSXR_XrFeatureStartInfo;
```

说明：

特性初始信息结构体

成员：

next 预留指针

featureType 指定特性类型

options 设置启用的特性功能选项

[函数] GSXR_StartXrFeature (初始信息运行)

```
GSXR_Result GSXR_StartXrFeature (
    GSXR_Runtime                          runtime,
    const GSXR_XrFeatureStartInfo*        startInfo,
    GSXR_XrFeature*                       feature);
```

说明：

起始特性运行

参数：

[in] runtime GSXR_Runtime 句柄，由 GSXR_CreateRuntime 获取

[in] startInfo 指向 GSXR_XrFeatureStartInfo 结构体，内含特性初始信息

[out] feature 特性初始化成功输出一个有效句柄，初始化失败输出

GSXR_NULL_HANDLE

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

备注：

特性状态需处于 GSXR_EventType_FeatureState_Available 才能起始特性运行

[基础能力规范] GSXR_StartXrFeature

[参数检验]

- `runtime` 必须是有效的 GSXR_Runtime 句柄。
- `startInfo` 必须是指向 GSXR_XrFeatureStartInfo 结构的有效指针。
- `feature` 必须是指向 GSXR_XrFeature 数值的有效指针。

[函数实现]

- Runtime 根据 startInfo 信息初始特性运行，并将特性句柄填入 feature 输出。

[返回值]

成功

- `GSXR_Result_Success` 特性起始成功。

失败

- `GSXR_Error_Operation_Failed` 特性起始失败。
- `GSXR_Error_Function_Unsupported` startInfo 的功能选项无法被完全支持。
- `GSXR_Error_Feature_Unsupported` 此设备不支持 startInfo 的 featureType 特性。
- `GSXR_Error_Handle_Invalid` runtime 为无效句柄。
- `GSXR_Error_Argument_Invalid` startInfo 或 feature 为 nullptr。或 startInfo 内容为无效信息。
- `GSXR_Error_Out_Of_Memory` 内存溢出。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

[函数] GSXR_StopXrFeature (停止特性运行)

```
GSXR_Result GSXR_StopXrFeature (
    GSXR_XrFeature              feature);
```

说明：

停止特性运行

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Runtime_Unavailable

备注：

无

[基础能力规范] GSXR_StopXrFeature

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。

[函数实现]

- Runtime 停止特性 feature 运行。

[返回值]

成功

- `GSXR_Result_Success` 特性运行停止成功。

失败

- `GSXR_Error_Operation_Failed` 特性运行停止失败。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。

11.1.5. 特性的属性信息

应用程序若需获取特性的属性信息，可调用 GSXR_GetXrFeatureProperties 获取特性的硬件及算法信息，若特性不具备该属性，字符串可为空值。

[结构体] GSXR_XrFeatureProperties (特性属性信息)

```
typedef struct GSXR_XrFeatureProperties {
    void*       next;
    char        serialNumber[GSXR_COMMON_STRING_MAX_SIZE];
    char        firmwareVersion[GSXR_COMMON_STRING_MAX_SIZE];
    char        vendorName[GSXR_COMMON_STRING_MAX_SIZE];
    char        algorithmVersion[GSXR_COMMON_STRING_MAX_SIZE];
    char        algorithmName[GSXR_COMMON_STRING_MAX_SIZE];
} GSXR_XrFeatureProperties;
```

说明：

特性属性结构体，用以描述特性的硬件及算法信息

成员：

next 预留指针

serialNumber 特性设备 SN 串号

firmwareVersion 特性设备 firmware 版本号

vendorName 特性厂家名称

algorithmVersion 特性算法版本号

algorithmName 特性算法名称

[函数] GSXR_GetXrFeatureProperties (获取特性属性)

```
GSXR_Result GSXR_GetXrFeatureProperties (
    GSXR_XrFeature                  feature,
    GSXR_XrFeatureProperties*       properties);
```

说明：

获取特性属性

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[out] properties 指向 GSXR_XrFeatureProperties 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetXrFeatureProperties

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。

- `properties` 必须是指向 GSXR_XrFeatureProperties 结构的有效指针。

[函数实现]

- XR 特性将自身属性信息填入 properties 中输出。

[返回值]

成功

- `GSXR_Result_Success` 特性属性信息获取成功。

失败

- `GSXR_Error_Operation_Failed` 特性属性信息获取失败。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` properties 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.2. XR 特性 - 眼球追踪

XR 眼球追踪特性只适用于支持眼球追踪的设备，通常运用于两类功能：

1. **实体眼球追踪**：用于追踪眼睛及眼球实体信息，如：眼球位姿、瞳孔直径、眼球焦点位置、眼睛开合度等数据，通常应用于眼球的特殊需求应用或虚拟场景中人体虚拟化的头像眼球表现。
2. **注视点追踪**：用于追踪双眼于世界坐标空间中的注视点位姿，通常应用于与内容中的场景注视交互。

[枚举] GSXR_EyeTrackingOptions (眼球追踪功能选项)

名称	数值	描述
GSXR_EyeTrackingOptions_Eye_Left	0x00000001	左眼追踪
GSXR_EyeTrackingOptions_Eye_Right	0x00000002	右眼追踪
GSXR_EyeTrackingOptions_Eye_Motion	0x00010000	实体眼球追踪
GSXR_EyeTrackingOptions_Gaze_Interaction	0x00020000	注视点追踪

11.2.1. 实体眼球数据

实体眼球数据区分为以下类型：

1. **眼球位姿**：用以描述眼球的位置及方向，使用本地坐标系，以两眼间的中心点面向脸部正前方为原点。
2. **瞳孔直径**：用以描述眼球瞳孔的直径大小，单位为毫米(mm)。
3. **眼球焦点位置**：归一化数据，用以描述以眼睛正中央为原点的眼球平面位置(x, y)，数据范围介于-1 到 1 之间。
4. **眼睛开合度**：归一化数据，用以描述眼睛的开合大小程度，数据范围介于 0 到 1 之间，0 为闭眼。

[枚举] GSXR_EyeFactor (眼球数据类型)

名称	数值	描述
GSXR_EyeFactor_Pose	0x00000001	眼球位姿
GSXR_EyeFactor_Pupil	0x00000002	瞳孔数据
GSXR_EyeFactor_Focal	0x00000004	眼球焦点位置
GSXR_EyeFactor_Openness	0x00000008	眼睛开合度

[结构体] GSXR_EyePose (眼球位姿)

```
typedef struct GSXR_EyePose {
   void*              next;
   GSXR_Bool32        isValid;
   GSXR_Posef         pose;
} GSXR_EyePose;
```

说明：

眼球位姿结构体

成员：

next 预留指针

isValid GSXR_TRUE 表示位姿有效，GSXR_FALSE 表示位姿无效

pose 眼球位姿（本地坐标系，以两眼间的中心点面向正前方为原点）

[结构体] GSXR_EyePupil（瞳孔数据）

```
typedef struct GSXR_EyePupil {
   void*              next;
   GSXR_Bool32        isValid;
   float              diameter;
} GSXR_EyePupil;
```

说明：

瞳孔数据结构体

成员：

next 预留指针

isValid GSXR_TRUE 表示直径有效，GSXR_FALSE 表示直径无效

diameter 瞳孔直径，单位为毫米 mm

[结构体] GSXR_EyeFocal（眼球焦点位置）

```
typedef struct GSXR_EyeFocal {
   void*              next;
   GSXR_Bool32        isValid;
   GSXR_Vector2f      point;
} GSXR_EyeFocal;
```

说明：

眼球焦点位置结构体（归一化数据）

成员：

next 预留指针

isValid GSXR_TRUE 表示直径有效，GSXR_FALSE 表示直径无效

point 眼球焦点位置坐标，数据范围为 -1 <= x <= 1，-1 <= y <= 1

[结构体] GSXE_EyeOpenness (眼睛开合度)

```
typedef struct GSXE_EyeOpenness {
    void*               next;
    GSXR_Bool32         isValid;
    float               openness;
} GSXE_EyeOpenness;
```

说明：

眼睛开合度结构体 (归一化数据)

成员：

next 预留指针

isValid GSXR_TRUE 表示开合度有效, GSXR_FALSE 表示开合度无效

openness 眼睛开合度, 数据范围为 0 - 1, 0 表示闭眼

[结构体] GSXR_EyeEntityData (眼球数据)

```
typedef struct GSXR_EyeEntityData {
    void*                 next;
    GSXR_Bool32           onTracking;
    GSXR_EyePose          pose;
    GSXR_EyePupil         pupil;
    GSXR_EyeFocal         focal;
    GSXE_EyeOpenness      openness;
} GSXR_EyeEntityData;
```

说明：

眼球数据结构体, 用以描述眼球的实体数据

成员：

next 预留指针

onTracking GSXR_TRUE 表示眼球追踪中, GSXR_FALSE 表示未起始眼球追踪

pose 眼球位姿

pupil 瞳孔数据

focal 眼球焦点位置

openness 眼睛开合度

鉴于各 XR 眼球追踪算法所支持的实体眼球追踪能力不一，且较多的追踪数据将耗用较多的运算资源,XR 应用程序可使用 GSXR_GetSupportedEyeFactors 函数获取眼球追踪算法支持的眼球数据类型，并以 GSXR_SetEyeFactorRequest 函数设置 XR 应用程序需要使用的所有眼球数据类型,使 XR 眼球追踪算法在正确提供应用程序眼球数据的前提下，可根据 XR 应用程序的需求调节算法的复杂度，以节省系统运算资源。

[函数] GSXR_GetSupportedEyeFactors (获取支持的眼球数据类型)

```
GSXR_Result GSXR_GetSupportedEyeFactors (
      GSXR_XrFeature         feature,
      GSXR_Flags64*          factors);
```

说明：

获取支持的眼球数据类型

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] factors GSXR_EyeFactor 的位掩码, 输出所有支持的眼球数据类型

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetSupportedEyeFactors

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `factors` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- 眼球追踪特性将支持的眼球数据类型进行位掩码的 OR 运算后填入 factors 中输出。

[返回值]

成功

- `GSXR_Result_Success` 眼球数据类型获取成功。

失败

- `GSXR_Error_Operation_Failed` 眼球数据类型获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持眼球追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` factors 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_SetEyeFactorRequest（设置使用的眼球数据类型）

```
GSXR_Result GSXR_SetEyeFactorRequest (
      GSXR_XrFeature         feature,
      GSXR_Flags64           request);
```

说明：

设置 XR 应用所有需要使用的眼球数据类型

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] request GSXR_EyeFactor 的位掩码，指定将要使用的眼球数据类型

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

XR 应用需优先调用 GSXR_SetEyeFactorRequest 完成眼球数据请求设置后，始能获取眼球数据

[基础能力规范] GSXR_SetEyeFactorRequest

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `request` 必须是有效的 GSXR_EyeFactor 位掩码。

[函数实现]

- 眼球追踪特性根据 request 的眼球数据类型请求进行追踪算法调节，并在 XR 应用程序调用 GSXR_GetEyeTrackingData 获取 GSXR_EyeTrackingData 时，输出 XR 应用请求的眼球数据类型数据。
- 在 XR 应用程序调用 GSXR_SetEyeFactorRequest 函数完成眼球数据请求设置前，XR 应用程序无法从 GSXR_EyeTrackingData 中获取实体眼球数据。

[返回值]

成功

- `GSXR_Result_Success` 眼球数据类型请求设置成功。

失败

- `GSXR_Error_Operation_Failed` 眼球数据类型请求设置失败。
- `GSXR_Error_Function_Unsupported` 眼球追踪特性不支持应用请求的眼球数据类型。
- `GSXR_Error_Feature_Unsupported` 不支持眼球追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` request 为无效的 GSXR_EyeFactor 位掩码。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.2.2. 注视点数据

注视点数据可用于 XR 应用程序中以眼球注视点位姿与内容场景物件进行交互，注视点数据的位姿以世界坐标系描述左右两眼瞳孔距离中心点位置及其注视方向。

[结构体] GSXE_EyeGaze (注视位姿)

```
typedef struct GSXR_EyeGaze {
   void*              next;
   GSXR_Bool32        isValid;
   GSXR_Posef         pose;
} GSXR_EyeGaze;
```

说明：

注视数据结构体 (世界坐标系)

成员：

next 预留指针

isValid GSXR_TRUE 表示位姿有效, GSXR_FALSE 表示位姿无效

pose 位姿 (世界坐标系)

[结构体] GSXR_EyeGazeData (注视数据)

```
typedef struct GSXR_EyeGazeData {
   void*              next;
   GSXR_Bool32        onTracking;
   GSXR_EyeGaze       gazePose;
} GSXR_EyeGazeData;
```

说明：

注视数据结构体

成员：

next 预留指针

onTracking GSXR_TRUE 表示注视点追踪中, GSXR_FALSE 表示未起始注视点追踪

gazePose 注视位姿

11.2.3. 眼球追踪数据

眼球追踪数据包含左、右眼的实体眼球数据及双眼的注视点数据，各数据中的 onTracking 成员代表此数据是否正由眼球特性算法追踪中，若 XR 应用程序在起始眼球追踪特性运行时未能使用该数据类型的功能选项、或尚未设置实体眼球数据类型的请求，则该数据结构中的 onTracking 成员即为 GSXR_FALSE。

[结构体] GSXR_EyeTrackingData (眼球追踪数据)

```
typedef struct GSXR_EyeTrackingData {
   void*                    next;
   GSXR_EyeEntityData       left;
   GSXR_EyeEntityData       right;
   GSXR_EyeGazeData         gaze;
} GSXR_EyeTrackingData;
```

说明：

眼球追踪数据结构体

成员：

next 预留指针

left 左眼球数据

right 右眼球数据

gaze 注视点数据

[函数] GSXR_GetEyeTrackingData (获取眼球追踪数据)

```
GSXR_Result GSXR_GetEyeTrackingData (
      GSXR_XrFeature              feature,
      GSXR_Time                   specifiedTime,
      GSXR_EyeTrackingData*       data);
```

说明：

获取指定时间的眼球追踪数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] specifiedTime 指定时间

[out] data 指向 GSXR_EyeTrackingData 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetEyeTrackingData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `specifiedTime` 必须是有效的时间戳数值。
- `data` 必须是指向 GSXR_EyeTrackingData 结构的有效指针。

[函数实现]

- 眼球追踪特性预测 specifiedTime 的眼球追踪数据填入 data 中输出。

[返回值]

成功

- `GSXR_Result_Success` 眼球追踪数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 眼球追踪数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持眼球追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` data 为 nullptr。或 specifiedTime 为一无效或算法无

法支持的时间戳数值。

- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.3. XR 特性 - 手部追踪

XR 手部追踪特性只适用于支持手部追踪的设备，通常运用于三类功能：

1. **手势识别**：用于识别手部追踪算法预先定义的手势类型，通常用于经手势识别触发手势事件并于应用程序中进行对应的内容行为。
2. **骨骼节点追踪**：用于追踪手部骨节在世界坐标空间中的位姿，通常用于将手部模型渲染在内容场景中。
3. **捏合姿态追踪**：用于追踪与大拇指形成捏合手势的手指与大拇指形成的捏合姿态数据，通常用于使用捏合姿态射线与内容中的场景物件进行交互，并以两指相互碰触捏合时触发点击事件。

[枚举] GSXR_HandTrackingOptions (手部追踪功能选项)

名称	数值	描述
GSXR_HandTrackingOptions_Hand_Left	0x00000001	左手追踪
GSXR_HandTrackingOptions_Hand_Right	0x00000002	右手追踪
GSXR_HandTrackingOptions_Gesture_Recognition	0x00010000	手势识别
GSXR_HandTrackingOptions_Skeleton_Motion	0x00020000	骨骼节点追踪
GSXR_HandTrackingOptions_Pinch_Interaction	0x00040000	捏合姿态追踪

11.3.1. 手势识别数据

基础手势类型定义于 GSXR_HandGestureType 中，GSXR_HandGestureType 所枚举的手势全部为单手手势，在一般状况下手势识别算法需分开识别左、右手的手势，并于 GSXR_HandGestureData 结构中分开描述。若手势算法延伸定义双手手势，则 GSXR_HandGestureData 中的左、右手手势需指定同一双手手势。

鉴于各 XR 手势识别算法可识别的手势类型不一，为获知 XR 应用可使用的手势，XR 应用程序可使用 GSXR_GetSupportedHandGestures 函数获取当前手部追踪特性可支持识别的手势类型，并以 GSXR_SetHandGestureRequest 函数设置 XR 应用程序所需求使用的所有手势类型，使 XR 手势识别算法依据 XR 应用需求在函数 GSXR_GetHandGestureData 中输出足够的手势识别数据。

[枚举] GSXR_HandGestureType (手势类型)

名称	数值	描述
GSXR_HandGestureType_Unavailable	0	未起始手势识别功能
GSXR_HandGestureType_Unknown	0x00000001	未知手势
GSXR_HandGestureType_Fist	0x00000002	握拳
GSXR_HandGestureType_Five	0x00000004	张开
GSXR_HandGestureType_OK	0x00000008	OK
GSXR_HandGestureType_ThumbUp	0x00000010	拇指朝上
GSXR_HandGestureType_IndexUp	0x00000020	食指朝上

[结构体] GSXR_HandGestureData (手势数据)

```
typedef struct GSXR_HandGestureData {
   void*                        next;
```

```
    GSXR_HandGestureType        left;
    GSXR_HandGestureType        right;
} GSXR_HandGestureData;
```

说明：

手势数据结构体

成员：

next 预留指针

left 左手手势

right 右手手势

[函数] GSXR_GetSupportedHandGestures (获取可识别的手势类型)

```
GSXR_Result GSXR_GetSupportedHandGestures (
    GSXR_XrFeature          feature,
    GSXR_Flags64*           gestures);
```

说明：

获取可识别的手势类型

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] gestures GSXR_HandGestureType 的位掩码, 输出所有支持的手势类型

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetSupportedHandGestures

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `gestures` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- 手部追踪特性将支持的手势类型进行位掩码的 OR 运算后填入 gestures 中输出。

[返回值]

成功

- `GSXR_Result_Success` 手势类型获取成功。

失败

- `GSXR_Error_Operation_Failed` 手势类型获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` gestures 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_SetHandGestureRequest (设置使用的手势类型)

```
GSXR_Result GSXR_SetHandGestureRequest (
      GSXR_XrFeature         feature,
      GSXR_Flags64           request);
```

说明：

设置 XR 应用所有需要使用的手势类型

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[in] request GSXR_HandGestureType 的位掩码, 指定将要使用的手势类型

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

XR 应用需优先调用 GSXR_SetHandGestureRequest 完成手势请求设置后，始能获取手势数据

[基础能力规范] GSXR_SetHandGestureRequest

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `request` 必须是有效的 GSXR_HandGestureType 位掩码。

[函数实现]

- 手部追踪特性根据 request 的手势类型请求进行识别算法调节，并在 XR 应用程序调用 GSXR_GetHandGestureData 获取 GSXR_HandGestureData 时，输出 XR 应用请求手势类型的识别数据。
- 在 XR 应用程序调用 GSXR_SetHandGestureRequest 函数完成手势类型请求设置前，XR 应用程序无法从 GSXR_HandGestureData 中获取手势识别数据。

[返回值]

成功

- `GSXR_Result_Success` 手势类型请求设置成功。

失败

- `GSXR_Error_Operation_Failed` 手势类型请求设置失败。
- `GSXR_Error_Function_Unsupported` 手部追踪特性不支持应用请求的手势类型。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` request 为无效的 GSXR_HandGestureType 位掩码。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_GetHandGestureData（获取手势数据）

```
GSXR_Result GSXR_GetHandGestureData (
      GSXR_XrFeature              feature,
      GSXR_HandGestureData*       data);
```

说明：

获取手势数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[out] data 指向 GSXR_HandGestureData 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

若识别为双手手势，left 与 right 指定同一双手手势

[基础能力规范] GSXR_GetHandGestureData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `data` 必须是指向 GSXR_HandGestureData 结构的有效指针。

[函数实现]

- 手部追踪特性识别手势类型填入 data 中输出。

[返回值]

成功

- `GSXR_Result_Success` 手势识别数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 手势识别数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。

- `GSXR_Error_Argument_Invalid` data 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

除调用 GSXR_GetHandGestureData 主动获取当前的手势数据外，XR 应用程序也可以通过使用 GSXR_RegisterHandGestureEventCallback 设置回调函数 GSXR_HandGestureEventCallback 获取手势事件 GSXR_HandGestureEvent，XR 应用程序设计回调函数必须考虑回调线程的即时性，不能在回调函数中执行过于耗时的内容处理工作。

[枚举] GSXR_Hand（左右手类型）

名称	数值	描述
GSXR_Hand_Left	0	左手
GSXR_Hand_Right	1	右手
GSXR_Hand_Both	2	双手

[结构体] GSXR_HandGestureEvent（手势识别事件）

```
typedef struct GSXR_HandGestureEvent {
    void*                   next;
    GSXR_Time               timestamp;
    GSXR_Hand               hand;
    GSXR_HandGestureType    gesture;
} GSXR_HandGestureEvent;
```

说明：

手势识别事件结构体

成员：

next 预留指针

timestamp 事件发生时间戳

hand 左,右,或双手

gesture 手势类型

[类型定义] GSXR_HandGestureEventCallback (手势识别事件回调函数指针)

```
typedef void (*GSXR_HandGestureEventCallback) (
    GSXR_XrFeature                      feature,
    const GSXR_HandGestureEvent*        event,
    void*                               cookie);
```

说明：

手势识别事件回调函数，当算法识别出应用需求的手势时回调

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] event 指向 GSXR_HandGestureEvent 结构体

[in] cookie 函数回调时带回的预备数值

返回值：

无

备注：

无

[函数] GSXR_RegisterHandGestureEventCallback (注册手势识别事件回调函数)

```
GSXR_Result GSXR_RegisterHandGestureEventCallback (
    GSXR_XrFeature                          feature,
    GSXR_HandGestureEventCallback           callback,
    void*                                   cookie);
```

说明：

注册手势识别事件回调函数

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] callback 手势识别事件回调函数

[in] cookie 函数回调时带回的预备数值

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Function_Unsupported

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_RegisterHandGestureEventCallback

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `callback` 必须是 GSXR_HandGestureEventCallback 函数类型的有效指针。

[函数实现]

- 手部追踪特性记录应用程序注册的 callback 函数指针，并于识别出应用设置需求的手势类型时回调 callback 函数。

[返回值]

成功

- `GSXR_Result_Success` 回调函数注册成功。

失败

- `GSXR_Error_Operation_Failed` 回调函数注册失败。
- `GSXR_Error_Function_Unsupported` 手部追踪特性不支持注册手势识别回调函数。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` callback 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.3.2. 骨骼节点数据

骨骼节点数据以手部 26 个骨节形成手部骨骼架构，各节点以其骨节名称在枚举 GSXR_HandJoint 中赋予命名，并可经由 GSXR_GetTrackedHandJoints 获取追踪算法支持追踪的手部节点。

```
#define GSXR_HAND_JOINT_COUNT 26
```

[枚举] GSXR_HandJoint（手部节点）

名称	数值	描述
GSXR_HandJoint_Palm	0	手掌节点
GSXR_HandJoint_Wrist	1	手腕节点
GSXR_HandJoint_Thumb_Metacarpal	2	大拇指掌骨节点
GSXR_HandJoint_Thumb_Proximal	3	大拇指近节指骨节点
GSXR_HandJoint_Thumb_Distal	4	大拇指中节指骨节点
GSXR_HandJoint_Thumb_Tip	5	大拇指指尖节点
GSXR_HandJoint_Index_Metacarpal	6	食指掌骨节点
GSXR_HandJoint_Index_Proximal	7	食指近节指骨节点
GSXR_HandJoint_Index_Intermediate	8	食指中节指骨节点
GSXR_HandJoint_Index_Distal	9	食指远节指骨节点
GSXR_HandJoint_Index_Tip	10	食指指尖节点
GSXR_HandJoint_Middle_Metacarpal	11	中指掌骨节点
GSXR_HandJoint_Middle_Proximal	12	中指近节指骨节点
GSXR_HandJoint_Middle_Intermediate	13	中指中节指骨节点
GSXR_HandJoint_Middle_Distal	14	中指远节指骨节点

GSXR_HandJoint_Middle_Tip	15	中指指尖节点
GSXR_HandJoint_Ring_Metacarpal	16	无名指掌骨节点
GSXR_HandJoint_Ring_Proximal	17	无名指近节指骨节点
GSXR_HandJoint_Ring_Intermediate	18	无名指中节指骨节点
GSXR_HandJoint_Ring_Distal	19	无名指远节指骨节点
GSXR_HandJoint_Ring_Tip	20	无名指指尖节点
GSXR_HandJoint_Little_Metacarpal	21	小指掌骨节点
GSXR_HandJoint_Little_Proximal	22	小指近节指骨节点
GSXR_HandJoint_Little_Intermediate	23	小指中节指骨节点
GSXR_HandJoint_Little_Distal	24	小指远节指骨节点
GSXR_HandJoint_Little_Tip	25	小指指尖节点

[函数] GSXR_GetTrackedHandJoints (获取可被追踪的手部节点)

```
GSXR_Result GSXR_GetTrackedHandJoints (
      GSXR_XrFeature         feature,
      GSXR_Flags64*          joints);
```

说明:

获取可被追踪的手部节点

参数:

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] joints (1 << GSXR_HandJoint) 的位掩码, 输出算法支持追踪的手部节点

返回值:

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetTrackedHandJoints

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `joints` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- 手部追踪特性将支持追踪的手部节点进行(1 << GSXR_HandJoint)位掩码的 OR 运算后填入 joints 中输出。

[返回值]

成功

- `GSXR_Result_Success` 可被追踪的手部节点获取成功。

失败

- `GSXR_Error_Operation_Failed` 可被追踪的手部节点获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` joints 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

各手部节点的姿态，根据手部追踪算法的能力差异，节点数据的有效性在不同设备上或有不同，节点姿态的位置、方向、线速度、角速度能力定义在枚举 GSXR_HandJointValidity 中，XR 应用程序可调用 GSXR_GetHandJointValidity 获取算法的跟踪能力位掩码，作为判别节点姿态数据内容有效性的依据。

[枚举] GSXR_HandJointValidity（手部节点空间追踪能力）

名称	数值	描述
GSXR_HandJointValidity_Pose_Position	0x00000001	追踪算法支持位置数据

GSXR_HandJointValidity_Pose_Orientation	0x00000002	追踪算法支持方向数据
GSXR_HandJointValidity_Velocity_Linear	0x00010000	追踪算法支持线速度数据
GSXR_HandJointValidity_Velocity_Angular	0x00020000	追踪算法支持角速度数据

[函数] GSXR_GetHandJointValidity (获取手部节点空间追踪能力)

```
GSXR_Result GSXR_GetHandJointValidity (
      GSXR_XrFeature         feature,
      GSXR_Flags64*          validity);
```

说明：

获取手部节点空间追踪能力

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] validity GSXR_HandJointValidity 的位掩码, 输出手部节点的追踪能力

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetHandJointValidity

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `validity` 必须是指向 GSXR_Flags64 数值的有效指针。

[函数实现]

- 手部追踪特性将手部节点追踪能力 GSXR_HandJointValidity 进行位掩码的 OR 运算后填

入 validity 中输出。

[返回值]

成功

- `GSXR_Result_Success` 手部节点追踪能力获取成功。

失败

- `GSXR_Error_Operation_Failed` 手部节点追踪能力获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` validity 为 nullptr。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

手部骨骼节点数据以 GSXR_HandSkeletonData 结构体表示，手部所有骨骼节点姿态数据以 GSXR_PoseState 的数组构成，数组长度为手部最大节点数目 26 (GSXR_HAND_JOINT_COUNT)。节点姿态以世界坐标表示，搭配头部视图姿态即可于场景中正确渲染手部模型。节点的本地坐标系为将双手手掌朝下平行地面，以 +x 朝右、+y 朝上、+z 朝后的轴向坐标构成。

[结构体] GSXR_HandSkeletonData (手部骨骼节点数据)

```
typedef struct GSXR_HandSkeletonData {
    void*                   next;
    GSXR_Bool32             onTracking;
    GSXR_Bool32             isValid;
    uint32_t                jointCount;
    GSXR_PoseState*         jointsPoseState;
} GSXR_HandSkeletonData;
```

说明：

手部骨骼节点数据结构体

成员：

next 预留指针

onTracking GSXR_TRUE 表示手部节点追踪中，GSXR_FALSE 表示未起始手部节点追

踪

isValid GSXR_TRUE 表示手部节点数据有效，GSXR_FALSE 表示手部节点数据无效

jointCount 手部节点数目，GSXR_HAND_JOINT_COUNT

jointsPoseState 指向 GSXR_PoseState 结构体的数组，输出手部节点姿态

11.3.3. 捏合姿态数据

捏合姿态由大拇指与其余四指中的任一指(通常为食指)形成捏合手势构成，通常用于内容场景中的物件交互(以捏合姿态的射线指向目标物件，再以捏合动作触发点击事件)，捏合数据以 GSXR_HandPinchData 结构体表示。

[枚举] GSXR_HandFinger (手指)

名称	数值	描述
GSXR_HandFinger_Thumb	1	大拇指
GSXR_HandFinger_Index	2	食指
GSXR_HandFinger_Middle	3	中指
GSXR_HandFinger_Ring	4	无名指
GSXR_HandFinger_Little	5	小指

[结构体] GSXR_HandPinchData (手部捏合数据)

```
typedef struct GSXR_HandPinchData {
    void*                   next;
    GSXR_Bool32             onTracking;
    GSXR_Bool32             isValid;
    GSXR_HandFinger         finger;
    float                   strength;
    GSXR_PoseState          pose;
} GSXR_HandPinchData;
```

说明：

手部捏合数据结构体

成员：

next 预留指针

onTracking GSXR_TRUE 表示手部捏合追踪中，GSXR_FALSE 表示未起始手部捏合追踪

isValid GSXR_TRUE 表示手部捏合数据有效，GSXR_FALSE 表示手部捏合数据无效

finger 往拇指捏合的手指

strength 捏合两指的临近度，数据范围为 0 - 1，1 表示两指相互碰触捏合

pose 捏合姿态

11.3.4. 手部追踪数据

结合上述骨骼节点数据及捏合姿态数据，构成的 GSXR_HandTrackingData 结构体可描述当前双手的节点姿态形成手部模型结构，以及捏合姿态的射线指向与捏合事件触发。各数据中的 onTracking 成员代表此数据是否正在被手部追踪特性算法追踪中，若 XR 应用程序在起始手部追踪特性运行时未使能该数据类型的功能选项，则该数据结构中的 onTracking 成员即为 GSXR_FALSE。

[结构体] GSXR_HandTrackingData（手部追踪数据）

```
typedef struct GSXR_HandTrackingData {
    void*                   next;
    GSXR_HandSkeletonData   leftHand;
    GSXR_HandSkeletonData   rightHand;
    GSXR_HandPinchData      leftPinch;
    GSXR_HandPinchData      rightPinch;
} GSXR_HandTrackingData;
```

说明：

手部追踪数据结构体

成员：

next 预留指针

leftHand 左手骨骼数据

rightHand 右手骨骼数据

leftPinch 左手捏合数据

rightPinch 右手捏合数据

[函数] GSXR_GetHandTrackingData（获取手部追踪数据）

```
GSXR_Result GSXR_GetHandTrackingData (
      GSXR_XrFeature                  feature,
      GSXR_Time                       specifiedTime,
      GSXR_HandTrackingData*          data);
```

说明：

获取指定时间的手部追踪数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] specifiedTime 指定时间

[out] data 指向 GSXR_HandTrackingData 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_GetHandTrackingData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `specifiedTime` 必须是有效的时间戳数值。
- `data` 必须是指向 GSXR_HandTrackingData 结构的有效指针。

[函数实现]

- 手部追踪特性预测 specifiedTime 的手部追踪数据填入 data 中输出。

[返回值]

成功

- `GSXR_Result_Success` 手部追踪数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 手部追踪数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持手部追踪特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` data 为 nullptr。或 specifiedTime 为一无效或算法无法支持的时间戳数值。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.4. XR 特性 - 语音命令

XR 语音命令特性只适用于支持语音命令的设备，设备所使用的语音引擎必须集成操作系统的多媒体系统，于 XR 应用程序起始语音命令输入动作时接收麦克风输入的语音进行语音命令识别，并依据既定的命令动作进行对应的行为处理。

[函数] GSXR_InputVoiceCommand (输入语音命令)

```
GSXR_Result GSXR_InputVoiceCommand (
    GSXR_XrFeature          feature,
    GSXR_Duration           timeout);
```

说明：

输入语音命令

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[in] timeout 语音输入的时限值, 单位为 naroseconds

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_InputVoiceCommand

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `timeout` 必须是有效的 GSXR_Duration 数值。

[函数实现]

- 语音命令特性开启麦克风接收语音命令输入，并持续接收至 timeout 后释放麦克风进行语音命令识别。
- timeout 值必须是一个语音引擎允许的合理时限值，不可过短或过长。

[返回值]

成功

- `GSXR_Result_Success` 成功开始接收语音命令输入。

失败

- `GSXR_Error_Operation_Failed` 无法开始接收语音命令输入。
- `GSXR_Error_Feature_Unsupported` 不支持语音命令特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` timeout 为一无效或算法无法支持的时限数值。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.5. XR 特性 - 手柄模型

XR 手柄模型特性只适用于支持手柄模型数据的设备。手柄模型数据描述于 GSXR_ControllerModelData 结构体中，模型结构由部件数组及纹理数组构成，部件数组用于描述模型结构的顶点及平面，纹理数组用于描述模型外观的像素图像，由此二种数据构成手柄模型。

[结构体] GSXR_ControllerModelData (手柄模型数据)

```
typedef struct GSXR_ControllerModelData {
    void*                       next;
    char                        name[GSXR_COMMON_STRING_MAX_SIZE];
    uint32_t                    componentCount;
    GSXR_ComponentInfo*         components;
    uint32_t                    textureCount;
    GSXR_TextureBitmap*         textures;
} GSXR_ControllerModelData;
```

说明：

手柄模型数据结构体

成员：

next 预留指针

name 手柄名称

componentCount 部件数目

components 部件信息数组

textureCount 纹理数目

textures 纹理缓存数组

部件信息描述于 GSXR_ComponentInfo 结构体中，手柄模型按设计可由一个或多个部件组成，每个部件各自描述其所有顶点的位置、法线和纹理对应位置，以及由顶点所形成的图元平面，对应的纹理图像索引，相对于手柄模型原点的部件位姿等信息，

[结构体] GSXR_ComponentInfo (部件信息)

```
typedef struct GSXR_ComponentInfo {
    void*                       next;
    char                        name[GSXR_COMMON_STRING_MAX_SIZE];
    GSXR_VertexBuffer           vertices;
    GSXR_VertexBuffer           normals;
    GSXR_VertexBuffer           texCoords;
    GSXR_IndexBuffer            indices;
    int32_t                     texIndex;
    float                       localMatrix[16];
    GSXR_Bool32                 defaultDraw;
} GSXR_ComponentInfo;
```

说明：

部件信息结构体

成员：

next 预留指针

name 部件名称

vertices 部件顶点位置信息 (顶点数目同 normals 及 texCoords, dimension 为 3)

normals 部件顶点法线信息 (顶点数目同 vertices 及 texCoords, dimension 为 3)

texCoords 部件纹理位置信息 (顶点数目同 vertices 及 normals, dimension 为 2)

indices 部件图元平面的顶点索引信息

texIndex 部件纹理于纹理缓存数组的索引值

localMatrix 部件模型原点相对于整体模型原点的本地端相对位姿

defaultDraw 是否预设绘制此部件

[结构体] GSXR_VertexBuffer (顶点缓存)

```
typedef struct GSXR_VertexBuffer {
    void*               next;
    float*              buffer;
    uint32_t            size;
    uint32_t            dimension;
```

```
} GSXR_VertexBuffer;
```

说明：

顶点缓存结构体

成员：

next 预留指针

buffer 顶点位置

size 顶点缓存大小

dimension 顶点维度，2 为平面（buffer 内容为 x，y，x，y，...），3 为立体（buffer 内容为 x，y，z，x，y，z，...）

[结构体] GSXR_IndexBuffer（图元缓存）

```
typedef struct GSXR_IndexBuffer {
   void*              next;
   uint32_t*          buffer;
   uint32_t           size;
   uint32_t           type;
} GSXR_IndexBuffer;
```

说明：

图元缓存结构体

成员：

next 预留指针

buffer 几何图元的顶点索引

size 几何图元缓存大小

type 几何图元类型及组成顶点数，1 为点（buffer 以一组索引为单位），2 为线（buffer 以两组索引为单位），3 为三角形（buffer 以三组索引为单位）

纹理信息描述于 GSXR_TextureBitmap 结构体中，包含其纹理 Bitmap 缓存，纹理宽度、高度，纹理像素格式等信息。

[结构体] GSXR_TextureBitmap（纹理 Bitmap 缓存）

```
typedef struct GSXR_TextureBitmap {
    void*               next;
    uint8_t*            buffer;
    uint32_t            width;
    uint32_t            height;
    int32_t             format;
    uint32_t            stride;
} GSXR_TextureBitmap;
```

说明：

纹理 Bitmap 缓存结构体

成员：

next 预留指针

buffer bitmap 缓存

width 纹理宽度，像素

height 纹理高度，像素

format 纹理像素格式（预设支持 RGBA_8888 格式，format 数值为 1）

stride 每一个 row 的位组大小（bytes）

XR 应用程序可调用 GSXR_AcquireControllerModelData 函数获取手柄模型的部件信息及纹理信息，并经由图形 API 渲染手柄模型。手柄模型数据使用完毕需调用 GSXR_ReleaseControllerModelData 释放内存。

[函数] GSXR_AcquireControllerModelData（获得手柄模型数据）

```
GSXR_Result GSXR_AcquireControllerModelData (
    GSXR_XrFeature                  feature,
    GSXR_ControllerModelData**      controllerModel);
```

说明：

获得手柄模型数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[out] controllerModel 指向 GSXR_ControllerModelData 指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

controllerModel 指针指向的 GSXR_ControllerModelData 指针指向手柄模型特性的 GSXR_ControllerModelData 结构，应用程序使用完毕后手柄模型数据必须调用 GSXR_ReleaseControllerModelData 释放内存

[基础能力规范] GSXR_AcquireControllerModelData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `controllerModel` 必须是指向 GSXR_ControllerModelData 指针的有效指针。

[函数实现]

- 手柄模型特性建构手柄模型数据，并将指向此数据内存的指针填入 controllerModel 中输出。

[返回值]

成功

- `GSXR_Result_Success` 手柄模型数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 手柄模型数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持手柄模型特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` controllerModel 为 nullptr。
- `GSXR_Error_Out_Of_Memory` 内存溢出。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_ReleaseControllerModelData（释放手柄模型数据）

```
GSXR_Result GSXR_ReleaseControllerModelData (
    GSXR_XrFeature                  feature,
    GSXR_ControllerModelData*       controllerModel);
```

说明：

释放手柄模型数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[out] controllerModel GSXR_ControllerModelData 指针，由 GSXR_AcquireControllerModelData 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_ReleaseControllerModelData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `controllerModel` 必须是指向 GSXR_ControllerModelData 结构的有效指针。

[函数实现]

- 手柄模型特性检验 controllerModel 指针确实为 GSXR_AcquireControllerModelData 输出的有效指针，并释放其内存。

[返回值]

成功

- `GSXR_Result_Success` 手柄模型数据释放成功。

失败

- `GSXR_Error_Operation_Failed` 手柄模型数据释放失败。
- `GSXR_Error_Feature_Unsupported` 不支持手柄模型特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` controllerModel 为无效指针。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_AcquireControllerModelData。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.6. XR 特性 - 点云数据

XR 点云特性只适用于支持即时运算点云数据的设备。点云数据描述于 GSXR_PointCloudData 结构体中，由一段连续的点集合数组构成，集合中的点个数 pointCount 必小于或等于点云算法所能支持的最大点个数 maxCount。点云数据中的各个点数据描述于 GSXR_PointData 结构体中，包含其点数据识别号 id 及点位置 position。

[结构体] GSXR_PointCloudData (点云数据)

```
typedef struct GSXR_PointCloudData {
    void*                       next;
    GSXR_Time                   timestamp;
    uint32_t                    maxCount;
    uint32_t                    pointCount;
    GSXR_PointData*             points;
} GSXR_PointCloudData;
```

说明：

点云数据结构体

成员：

next 预留指针

timestamp 点云数据时间戳

maxCount 点云算法支持的最大点个数

pointCount 点集合的点个数

points 点集合，点数据识别号 0 <= id < pointCount

[结构体] GSXR_PointData (点数据)

```
typedef struct GSXR_PointData {
    void*                       next;
    uint32_t                    id;
    GSXR_Vector3f               position;
} GSXR_PointData;
```

说明：

点数据结构体

成员：

next 预留指针

id 点数据识别号

position 位置

XR 应用程序可调用 GSXR_AcquirePointCloudData 函数获取点云数据。点云数据使用完毕后需调用 GSXR_ReleasePointCloudData 释放内存。

[函数] GSXR_AcquirePointCloudData (获得点云数据)

```
GSXR_Result GSXR_AcquirePointCloudData (
    GSXR_XrFeature                  feature,
    GSXR_PointCloudData**           pointCloud);
```

说明：

获得点云数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[out] pointCloud 指向 GSXR_PointCloudData 指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

pointCloud 指针指向的 GSXR_PointCloudData 指针指向点云特性的 GSXR_PointCloudData 结构，应用程序使用完毕后点云数据必须调用 GSXR_ReleasePointCloudData 释放点云内存

[基础能力规范] GSXR_AcquirePointCloudData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `pointCloud` 必须是指向 GSXR_PointCloudData 指针的有效指针。

[函数实现]

- 点云特性建构点云数据，并将指向此数据内存的指针填入 pointCloud 中输出。

[返回值]

成功

- `GSXR_Result_Success` 点云数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 点云数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持点云特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` pointCloud 为 nullptr。
- `GSXR_Error_Out_Of_Memory` 内存溢出。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_ReleasePointCloudData (释放点云数据)

```
GSXR_Result GSXR_ReleasePointCloudData (
    GSXR_XrFeature                  feature,
    GSXR_PointCloudData*            pointCloud);
```

说明：

释放点云数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] pointCloud GSXR_PointCloudData 指针, 由 GSXR_AcquirePointCloudData 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_ReleasePointCloudData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `pointCloud` 必须是指向 GSXR_PointCloudData 结构的有效指针。

[函数实现]

- 点云特性检验 pointCloud 指针确实为 GSXR_AcquirePointCloudData 输出的有效指针，并释放其内存。

[返回值]

成功

- `GSXR_Result_Success` 点云数据释放成功。

失败

- `GSXR_Error_Operation_Failed` 点云数据释放失败。
- `GSXR_Error_Feature_Unsupported` 不支持点云特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` pointCloud 为无效指针。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_AcquirePointCloudData。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.7. XR 特性 - 平面侦测

XR 平面侦测特性只适用于支持平面侦测的设备，通常用于侦测三类平面：

1. 平行地面正面朝下的平面，如天花板。
2. 平行地面正面朝上的平面，如地板、桌面。
3. 垂直地面的平面，如墙面。

XR 应用程序需根据内容需求的平面功能选项起始平面侦测特性，以节省应用程序不需要的平面类型的运算资源。

[枚举] GSXR_PlaneOptions (平面特性功能选项)

名称	数值	描述
GSXR_PlaneOptions_Horizontal_DownwardFacing	0x00000001	平行地面正面朝下的平面
GSXR_PlaneOptions_Horizontal_UpwardFacing	0x00000002	平行地面正面朝上的平面
GSXR_PlaneOptions_Vertical	0x00000004	垂直地面的平面

平面追踪数据描述于 GSXR_PlaneTrackerData 结构体中，由一段连续的平面集合数组构成，集合中的平面个数 planeCount 必小于或等于平面侦测算法所能支持的最大平面数 maxCount。

[结构体] GSXR_PlaneTrackerData（平面追踪数据）

```
typedef struct GSXR_PlaneTrackerData {
    void*                       next;
    GSXR_Time                   timestamp;
    uint32_t                    maxCount;
    uint32_t                    planeCount;
    GSXR_PlaneData*             planes;
} GSXR_PlaneTrackerData;
```

说明：

平面追踪数据结构体

成员：

next 预留指针

timestamp 平面数据时间戳

maxCount 平面算法支持的最大平面个数

planeCount 平面集合的平面个数

planes 平面集合，平面数据 0 <= id < planeCount

平面追踪数据中的各个平面数据描述于 GSXR_PlaneData 结构体中，包含其平面识别号 id、平面类型 type、平面中心点 center、延伸平面 extentX/extentZ、及凸面多边形平面顶点集合 vertices。

[结构体] GSXR_PlaneData（平面数据）

```
typedef struct GSXR_PlaneData {
    void*                       next;
    uint32_t                    id;
```

```
    GSXR_PlaneType              type;
    GSXR_Posef                  center;
    float                       extentX;
    float                       extentZ;
    uint32_t                    vertexCount;
    GSXR_PlaneVertexData*       vertices;
} GSXR_PlaneData;
```

说明：

平面数据结构体

成员：

next 预留指针

id 平面识别号

type 平面类型

center 平面中心点

extentX 以 center 为中心点的延伸平面 x 边长度，单位为米

extentZ 以 center 为中心点的延伸平面 z 边长度，单位为米

vertexCount 平面顶点集合的点个数

vertices 平面顶点集合，以 center 为中心点围绕二维凸面多边形平面

[枚举] GSXR_PlaneType（平面类型）

名称	数值	描述
GSXR_PlaneType_Invalid	0	无效的平面
GSXR_PlaneType_Horizontal_DownwardFacing	1	正面朝下的水平面，如天花板
GSXR_PlaneType_Horizontal_UpwardFacing	2	正面朝上的水平面，如地板,桌面
GSXR_PlaneType_Vertical	3	垂直平面，如墙面

[结构体] GSXR_PlaneVertexData（平面顶点数据）

```
typedef struct GSXR_PlaneVertexData {
    float       x;
    float       z;
```

```
} GSXR_PlaneVertexData;
```

说明：

平面顶点数据结构体

成员：

x 二维平面顶点 x 坐标

z 二维平面顶点 z 坐标

XR 应用程序可调用 GSXR_AcquirePlaneTrackerData 函数获取平面追踪数据。平面追踪数据使用完毕后会调用 GSXR_ReleasePlaneTrackerData 释放内存。

[函数] GSXR_AcquirePlaneTrackerData (获得平面数据)

```
GSXR_Result GSXR_AcquirePlaneTrackerData (
    GSXR_XrFeature                  feature,
    GSXR_PlaneTrackerData**         planeTracker);
```

说明：

获得平面数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] planeTracker 指向 GSXR_PlaneTrackerData 指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

planeTracker 指针指向的 GSXR_PlaneTrackerData 指针指向平面侦测特性的 GSXR_PlaneTrackerData 结构, 应用程序用毕平面数据必须调用

GSXR_ReleasePlaneTrackerData 释放平面内存

[基础能力规范] GSXR_AcquirePlaneTrackerData

[参数检验]

- feature 必须是有效的 GSXR_XrFeature 句柄。
- planeTracker 必须是指向 GSXR_PlaneTrackerData 指针的有效指针。

[函数实现]

- 平面侦测特性构建平面追踪数据，并将指向此数据内存的指针填入 planeTracker 中输出。

[返回值]

成功

- GSXR_Result_Success 平面追踪数据获取成功。

失败

- GSXR_Error_Operation_Failed 平面追踪数据获取失败。
- GSXR_Error_Feature_Unsupported 不支持平面侦测特性。
- GSXR_Error_Handle_Invalid feature 为无效句柄。
- GSXR_Error_Argument_Invalid planeTracker 为 nullptr。
- GSXR_Error_Out_Of_Memory 内存溢出。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Feature_Unavailable 特性未处于正确工作状态。

[函数] GSXR_ReleasePlaneTrackerData (释放平面数据)

```
GSXR_Result GSXR_ReleasePlaneTrackerData (
      GSXR_XrFeature                feature,
      GSXR_PlaneTrackerData*        planeTracker);
```

说明：

释放平面数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] planeTracker GSXR_PlaneTrackerData 指针, 由 GSXR_AcquirePlaneTrackerData 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_ReleasePlaneTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `planeTracker` 必须是指向 GSXR_PlaneTrackerData 结构的有效指针。

[函数实现]

- 平面侦测特性检验 planeTracker 指针确实为 GSXR_AcquirePlaneTrackerData 输出的有效指针，并释放其内存。

[返回值]

成功

- `GSXR_Result_Success` 平面追踪数据释放成功。

失败

- `GSXR_Error_Operation_Failed` 平面追踪数据释放失败。
- `GSXR_Error_Feature_Unsupported` 不支持平面侦测特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` planeTracker 为无效指针。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_AcquirePlaneTrackerData。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.8. XR 特性 - 图像识别

XR 图像识别特性只适用于支持图像识别算法的设备。XR 应用程序必须先加载欲识别的源图像并指定其图像区域，XR 图像识别特性将赋予加载成功的源图像一个有效的图像标识号，XR 应用程序可使用此源图像标识号要求 XR 图像识别特性比对目标图像的指定区域中是否包含此源图像，若识别成功则 XR 特性将在目标图像上标示图像标签，描述其识别出的图像区域及对应的源图像标识号。

11.8.1. 源图像的加载与卸载

XR 应用程序可调用 GSXR_LoadModelImage 函数加载源图像，图像的资源标识符需符合操作系统定义的格式，使 XR 图像识别特性可以正确加载图像资源，且应用程序加载源图像必须明确指定欲识别的图像区域，以供后续于目标图像上的识别对比。源图像加载成功可获取一个图像标识号 GSXR_ImageId，若识别算法许可，也可以加载多个源图像之后在一个目标图像上同时识别。当识别完毕时，XR 应用程序可调用 GSXR_UnloadModelImage 函数卸载源图像资源。

```
GSXR_DEFINE_ATOM(GSXR_ImageId)        图像标识号
```

[结构体] GSXR_ImageInfo (图像信息)

```
typedef struct GSXR_ImageInfo {
   void*                next;
   const char*          uri;
   int32_t              left;
   int32_t              top;
   int32_t              width;
   int32_t              height;
```

```
} GSXR_ImageInfo;
```

说明：

图像内容结构体

成员：

next 预留指针

uri 图像资源标识符

left 图像识别区域起始点 x 轴位置，单位为像素

top 图像识别区域起始点 y 轴位置，单位为像素

width 图像识别区域宽度，单位为像素

height 图像识别区域高度，单位为像素

[函数] GSXR_LoadModelImage (加载欲识别的模型图像)

```
GSXR_Result GSXR_LoadModelImage (
    GSXR_XrFeature                  feature,
    const GSXR_ImageInfo*           imageInfo,
    GSXR_ImageId*                   modelId);
```

说明：

加载欲识别的模型图像

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] imageInfo 指向 GSXR_ImageInfo 结构体，指定模型图像内容

[out] modelId 若模型图像加载成功，输出模型图像标识号

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_LoadModelImage

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `imageInfo` 必须是指向 GSXR_ImageInfo 结构的有效指针。
- `modelId` 必须是指向 GSXR_ImageId 数值的有效指针。

[函数实现]

- 图像识别特性根据 imageInfo 的图像资源标识符加载模型图像并读取其指定图像区域，加载完成后生成一图像标识号填入 modelId 中输出。

[返回值]

成功

- `GSXR_Result_Success` 模型图像加载成功。

失败

- `GSXR_Error_Operation_Failed` 模型图像加载失败。
- `GSXR_Error_Feature_Unsupported` 不支持图像识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` imageInfo 为 nullptr。或 modelId 为 nullptr。
- `GSXR_Error_Out_Of_Memory` 内存溢出。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_UnloadModelImage (卸载模型图像)

```
GSXR_Result GSXR_UnloadModelImage (
      GSXR_XrFeature          feature,
      GSXR_ImageId            modelId);
```

说明：

卸载模型图像

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[in] modelId GSXR_ImageId 标识号, 由 GSXR_LoadModelImage 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_UnloadModelImage

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `modelId` 必须是有效的图像标识号 GSXR_ImageId。

[函数实现]

- 图像识别特性卸载模型图像并释放资源。

[返回值]

成功

- `GSXR_Result_Success` 模型图像卸载成功。

失败

- `GSXR_Error_Operation_Failed` 模型图像卸载失败。
- `GSXR_Error_Feature_Unsupported` 不支持图像识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` modelId 为无效的图像标识号。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_LoadModelImage。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.8.2. 识别目标图像

在源图像加载成功之后，XR 应用可构建 GSXR_ImageTrackerAcquireInfo 描述图像识别的请求，其中包含其目标图像信息、源图像标识号、及识别算法的最小置信度，并调用 GSXR_AcquireImageTrackerData 函数进行图像识别，图像识别结果数据描述存放在 GSXR_ImageTrackerData 结构中，对于识别到的图像内容以 GSXR_ImageMatchData 结构描述，其中包含其目标图像上的图像标签、对应的源图像标识号、以及识别置信度，XR 应用程序可根据内容需求使用这些图像识别数据。在图像识别数据使用完毕后，需调用 GSXR_ReleaseImageTrackerData 函数释放图像识别数据。

[结构体] GSXR_ImageTrackerAcquireInfo (图像识别请求)

```
typedef struct GSXR_ImageTrackerAcquireInfo {
   void*                        next;
   const GSXR_ImageInfo*        targetImageInfo;
   uint32_t                     sourceImageCount;
   const GSXR_ImageId*          sourceImages;
   float                        minConfidence;
} GSXR_ImageTrackerAcquireInfo;
```

说明：

图像识别请求信息结构体

成员：

next 预留指针

targetImageInfo 待检目标图像

sourceImageCount 欲识别的源图像数目

sourceImages 欲识别的源图像数组

minConfidence 设置识别结果的最小识别置信度，介于 0 - 1

[结构体] GSXR_ImageLabel (图像标签)

```
typedef struct GSXR_ImageLabel {
   void*                        next;
   GSXR_ImageId                 imageId;
   int32_t                      left;
   int32_t                      top;
   int32_t                      width;
   int32_t                      height;
} GSXR_ImageLabel;
```

说明：

图像标签结构体

成员：

next 预留指针

imageId 目标图像标识号

left 标签区域起始点 x 轴位置，单位为像素

top 标签区域起始点 y 轴位置，单位为像素

width 标签区域宽度，单位为像素

height 标签区域高度，单位为像素

[结构体] GSXR_ImageMatchData（图像匹配数据）

```
typedef struct GSXR_ImageMatchData {
   void*                        next;
   GSXR_ImageLabel              target;
   GSXR_ImageId                 source;
   float                        confidence;
} GSXR_ImageMatchData;
```

说明：

图像匹配数据结构体

成员：

next 预留指针

target 识别出的目标图像标签

source 识别出的对应源图像标识号

confidence 识别置信度，介于 0 - 1

[结构体] GSXR_ImageTrackerData (图像识别结果数据)

```
typedef struct GSXR_ImageTrackerData {
   void*                        next;
   uint32_t                     matchCount;
   GSXR_ImageMatchData*         matches;
} GSXR_ImageTrackerData;
```

说明：

图像识别结果数据结构体

成员：

next 预留指针

matchCount 在目标图像上识别出的图像个数

matches 识别出的图像匹配信息

[函数] GSXR_AcquireImageTrackerData (获得图像识别结果数据)

```
GSXR_Result GSXR_AcquireImageTrackerData (
      GSXR_XrFeature                          feature,
      const GSXR_ImageTrackerAcquireInfo*     acquireInfo,
      GSXR_ImageTrackerData**                 imageTracker);
```

说明：

获得图像识别结果数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[in] acquireInfo 指向 GSXR_ImageTrackerAcquireInfo 结构体, 指定图像识别请求信息

[out] imageTracker 指向 GSXR_ImageTrackerData 指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

imageTracker 指针指向的 GSXR_ImageTrackerData 指针指向图像识别特性的 GSXR_ImageTrackerData 结构，应用程序用毕图像识别结果数据必须调用 GSXR_ReleaseImageTrackerData 释放内存

[基础能力规范] GSXR_AcquireImageTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `acquireInfo` 必须是指向 GSXR_ImageTrackerAcquireInfo 结构的有效指针。
- `imageTracker` 必须是指向 GSXR_ImageTrackerData 指针的有效指针。

[函数实现]

- 图像识别特性根据 acquireInfo 的请求信息建构图像识别结果数据，并将指向此数据内存的指针填入 imageTracker 中输出。

[返回值]

成功

- `GSXR_Result_Success` 图像识别结果数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 图像识别结果数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持图像识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` acquireInfo 为 nullptr。或 imageTracker 为 nullptr。
- `GSXR_Error_Out_Of_Memory` 内存溢出。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_ReleaseImageTrackerData (释放图像识别结果数据)

```
GSXR_Result GSXR_ReleaseImageTrackerData (
```

```
    GSXR_XrFeature                  feature,
    GSXR_ImageTrackerData*          imageTracker);
```

说明：

释放图像识别结果数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[out] imageTracker GSXR_ImageTrackerData 指针，由 GSXR_AcquireImageTrackerData 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_ReleaseImageTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `imageTracker` 必须是指向 GSXR_ImageTrackerData 结构的有效指针。

[函数实现]

- 图像识别特性检验 imageTracker 指针确实为 GSXR_AcquireImageTrackerData 输出的有效指针，并释放其内存。

[返回值]

成功

- `GSXR_Result_Success` 图像识别结果数据释放成功。

失败

- `GSXR_Error_Operation_Failed` 图像识别结果数据释放失败。

- `GSXR_Error_Feature_Unsupported` 不支持图像识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` imageTracker 为无效指针。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_AcquireImageTrackerData。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.9. XR 特性 - 脸部识别

XR 脸部识别特性只适用于支持脸部识别算法的设备，通常运用于三类功能：

1. **识别人脸：** 从目标图像中根据人脸构造识别出人脸。
2. **比对人脸：** 使用已识别的人脸比对目标图像中是否有相同特征的人脸。
3. **提取脸部特征：**从已识别的人脸中提取每个特征点位置。

11.9.1. 识别人脸

XR 应用程序必须先输入欲识别的目标图像 GSXR_FaceImageInfo，包含其资源标识符(图像的资源标识符需符合操作系统定义的格式，使 XR 脸部识别特性可以正确加载图像资源)及指定图像中的识别区域，并以此构建脸部识别的请求 GSXR_FaceTrackerAcquireInfo 后调用 GSXR_AcquireFaceTrackerData 进行脸部识别，XR 脸部识别特性将从指定的图像区域中识别是否存在人脸，识别出的人脸将被赋予一个脸部标签 GSXR_FaceLabel，包含一个有效的脸部标识号、标签区域、及其脸部识别置信度。在脸部识别数据使用完毕之后，XR 应用程序需调用 GSXR_ReleaseFaceTrackerData 释放资源。

```
GSXR_DEFINE_ATOM(GSXR_FaceId)        脸部标识号
```

[结构体] GSXR_FaceImageInfo (脸部图像信息)

```
typedef struct GSXR_FaceImageInfo {
    void*                next;
    const char*          uri;
    int32_t              left;
    int32_t              top;
    int32_t              width;
    int32_t              height;
} GSXR_FaceImageInfo;
```

说明：

脸部图像内容结构体

成员：

next 预留指针

uri 脸部图像资源标识符

left 脸部图像识别区域起始点 x 轴位置，单位为像素

top 脸部图像识别区域起始点 y 轴位置，单位为像素

width 脸部图像识别区域宽度，单位为像素

height 脸部图像识别区域高度，单位为像素

[结构体] GSXR_FaceTrackerAcquireInfo (脸部识别请求信息)

```
typedef struct GSXR_FaceTrackerAcquireInfo {
    void*                               next;
    const GSXR_FaceImageInfo*           targetImageInfo;
    float                               minConfidence;
} GSXR_FaceTrackerAcquireInfo;
```

说明：

脸部识别请求内容结构体

成员：

next 预留指针

targetImageInfo 待检脸部目标图像

minConfidence 设置识别结果的最小识别置信度，介于 0 - 1

[结构体] GSXR_FaceLabel (脸部标签)

```
typedef struct GSXR_FaceLabel {
    void*                next;
    GSXR_FaceId          faceId;
    int32_t              left;
    int32_t              top;
    int32_t              width;
    int32_t              height;
    float                confidence;
} GSXR_FaceLabel;
```

说明：

脸部标签结构体

成员：

next 预留指针

faceId 脸部标识号

left 标签区域起始点 x 轴位置，单位为像素

top 标签区域起始点 y 轴位置，单位为像素

width 标签区域宽度，单位为像素

height 标签区域高度，单位为像素

confidence 脸部识别置信度，介于 0 - 1

[结构体] GSXR_FaceTrackerData (脸部识别结果数据)

```
typedef struct GSXR_FaceTrackerData {
    void*                next;
    GSXR_Time            timestamp;
    uint32_t             labelCount;
    GSXR_FaceLabel*      labels;
} GSXR_FaceTrackerData;
```

说明：

脸部识别结果数据结构体

成员：

next 预留指针

timestamp 时间戳

labelCount 在图像上识别出的脸部标签个数

labels 脸部标签信息

[函数] GSXR_AcquireFaceTrackerData (获得脸部识别结果数据)

```
GSXR_Result GSXR_AcquireFaceTrackerData (
    GSXR_XrFeature                          feature,
    const GSXR_FaceTrackerAcquireInfo*      acquireInfo,
    GSXR_FaceTrackerData**                  faceTracker);
```

说明：

获得脸部识别结果数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[in] acquireInfo 指向 GSXR_FaceTrackerAcquireInfo 结构体, 指定脸部识别请求信息

[out] faceTracker 指向 GSXR_FaceTrackerData 指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

faceTracker 指针指向的 GSXR_FaceTrackerData 指针指向脸部识别特性的 GSXR_FaceTrackerData 结构, 应用程序用毕脸部识别结果数据必须调用 GSXR_ReleaseFaceTrackerData 释放内存

[基础能力规范] GSXR_AcquireFaceTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `acquireInfo` 必须是指向 GSXR_FaceTrackerAcquireInfo 结构的有效指针。
- `faceTracker` 必须是指向 GSXR_FaceTrackerData 指针的有效指针。

[函数实现]

- 脸部识别特性根据 acquireInfo 请求构建脸部识别结果数据，并将指向此数据内存的指针填入 faceTracker 中输出。

[返回值]

成功

- `GSXR_Result_Success` 脸部识别结果数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 脸部识别结果数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持脸部识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` acquireInfo 为 nullptr。或 faceTracker 为 nullptr。
- `GSXR_Error_Out_Of_Memory` 内存溢出。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

[函数] GSXR_ReleaseFaceTrackerData (释放脸部识别结果数据)

```
GSXR_Result GSXR_ReleaseFaceTrackerData (
      GSXR_XrFeature                  feature,
      GSXR_FaceTrackerData*           faceTracker);
```

说明：

释放脸部识别结果数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] faceTracker GSXR_FaceTrackerData 指针, 由 GSXR_AcquireFaceTrackerData 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_ReleaseFaceTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `faceTracker` 必须是指向 GSXR_FaceTrackerData 结构的有效指针。

[函数实现]

- 脸部识别特性检验 faceTracker 指针确实为 GSXR_AcquireFaceTrackerData 输出的有效指针，并释放其内存。

[返回值]

成功

- `GSXR_Result_Success` 脸部识别结果数据释放成功。

失败

- `GSXR_Error_Operation_Failed` 脸部识别结果数据释放失败。
- `GSXR_Error_Feature_Unsupported` 不支持脸部识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` faceTracker 为无效指针。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_AcquireFaceTrackerData。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.9.2. 比对人脸

在识别出脸部标签后，若要在另外的目标图像上比对是否存在相同的人脸，XR 应用程序可构建脸部比对请求 GSXR_FaceMatchTrackerAcquireInfo，包含待比对的目标图像信息、欲识别的源脸部标签、以及最小识别相似度，调用函数 GSXR_AcquireFaceMatchTrackerData 进行人脸比对，XR 脸部识别特性将比对目标图像的指定区域中是否包含具备此脸部标签特征的人脸，若识别成功则 XR 脸部识别特性将于目标图像上标示脸部标签 GSXR_FaceLabel，描述其识别出的脸部图像区域。XR 应用程序在使用完脸部比对结果数据后，必须调用 GSXR_ReleaseFaceMatchTrackerData 释放资源。

[结构体] GSXR_FaceMatchTrackerAcquireInfo（脸部比对请求信息）

```
typedef struct GSXR_FaceMatchTrackerAcquireInfo {
    void*                               next;
    const GSXR_FaceImageInfo*           targetImageInfo;
    uint32_t                            sourceFaceCount;
    const GSXR_FaceLabel*               sourceFaces;
    float                               minSimilarity;
} GSXR_FaceMatchTrackerAcquireInfo;
```

说明：

脸部比对请求信息结构体

成员：

next 预留指针

targetImageInfo 待检目标图像

sourceFaceCount 欲比对的源脸部数目

sourceFaces 欲比对的源脸部数组

minSimilarity 设置识别结果的最小识别相似度，介于 0 - 1

[结构体] GSXR_FaceMatchData（脸部比对数据）

```
typedef struct GSXR_FaceMatchData {
   void*                next;
   GSXR_FaceLabel       target;
   GSXR_FaceLabel       source;
   float                similarity;
} GSXR_FaceMatchData;
```

说明：

脸部比对数据结构体

成员：

next 预留指针

target 识别出的目标脸部标签

source 识别出的对应源脸部标签

similarity 脸部相似度，介于 0 - 1

[结构体] GSXR_FaceMatchTrackerData（脸部比对结果数据）

```
typedef struct GSXR_FaceMatchTrackerData {
   void*                    next;
   GSXR_Time                timestamp;
   uint32_t                 matchCount;
   GSXR_FaceMatchData*      matches;
} GSXR_FaceMatchTrackerData;
```

说明：

脸部比对结果数据结构体

成员：

next 预留指针

timestamp 时间戳

matchCount 在目标图像上识别出的脸部个数

matches 识别的脸部信息

[函数] GSXR_AcquireFaceMatchTrackerData（获得脸部比对结果数据）

```
GSXR_Result GSXR_AcquireFaceMatchTrackerData (
```

```
    GSXR_XrFeature                                  feature,
    const GSXR_FaceMatchTrackerAcquireInfo*         acquireInfo,
    GSXR_FaceMatchTrackerData**                     faceMatchTracker);
```

说明：

获得脸部比对结果数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] acquireInfo 指向 GSXR_FaceMatchTrackerAcquireInfo 结构体，指定脸部比对请求信息

[out] faceMatchTracker 指向 GSXR_FaceMatchTrackerData 指针

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Out_Of_Memory

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

faceMatchTracker 指针指向的 GSXR_FaceMatchTrackerData 指针指向脸部识别特性的 GSXR_FaceMatchTrackerData 结构，应用程序使用完毕后脸部比对结果数据必须调用 GSXR_ReleaseFaceMatchTrackerData 释放内存

[基础能力规范] GSXR_AcquireFaceMatchTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `acquireInfo` 必须是指向 GSXR_FaceMatchTrackerAcquireInfo 结构的有效指针。
- `faceMatchTracker` 必须是指向 GSXR_FaceMatchTrackerData 指针的有效指针。

[函数实现]

- 脸部识别特性根据 acquireInfo 的请求信息建构脸部比对结果数据，并将指向此数据内存的指针填入 faceMatchTracker 中输出。

[返回值]

成功

- GSXR_Result_Success 脸部比对结果数据获取成功。

失败

- GSXR_Error_Operation_Failed 脸部比对结果数据获取失败。
- GSXR_Error_Feature_Unsupported 不支持脸部识别特性。
- GSXR_Error_Handle_Invalid feature 为无效句柄。
- GSXR_Error_Argument_Invalid acquireInfo 为 nullptr。或 faceMatchTracker 为 nullptr。
- GSXR_Error_Out_Of_Memory 内存溢出。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Feature_Unavailable 特性未处于正确工作状态。

[函数] GSXR_ReleaseFaceMatchTrackerData (释放脸部比对结果数据)

```
GSXR_Result GSXR_ReleaseFaceMatchTrackerData (
      GSXR_XrFeature                  feature,
      GSXR_FaceMatchTrackerData *     faceMatchTracker);
```

说明：

释放脸部比对结果数据

参数：

[in] feature GSXR_XrFeature 句柄, 由 GSXR_StartXrFeature 获取

[out] faceMatchTracker GSXR_FaceMatchTrackerData 指针, 由 GSXR_AcquireFaceMatchTrackerData 获取

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Call_Flow_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

无

[基础能力规范] GSXR_ReleaseFaceMatchTrackerData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `faceMatchTracker` 必须是指向 GSXR_FaceMatchTrackerData 结构的有效指针。

[函数实现]

- 脸部识别特性检验 faceMatchTracker 指针确实为 GSXR_AcquireFaceMatchTrackerData 输出的有效指针，并释放其内存。

[返回值]

成功

- `GSXR_Result_Success` 脸部比对结果数据释放成功。

失败

- `GSXR_Error_Operation_Failed` 脸部比对结果数据释放失败。
- `GSXR_Error_Feature_Unsupported` 不支持脸部识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` faceMatchTracker 为无效指针。
- `GSXR_Error_Call_Flow_Invalid` 未先调用 GSXR_AcquireFaceMatchTrackerData。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

11.9.3. 提取脸部特征

人脸图像的脸部特征可区分为以下部位,各部位有其特征点且个数不一，脸部特征的提取便是描述脸部标签中各部位的特征点位置。

1. **眉毛特征点**：由 5 个特征点组成，定义如下。

```
#define GSXR_EYEBROW_FEATURE_COUNT 5
```

[枚举] GSXR_EyebrowFeature (眉毛特征点)

名称	数值	描述
GSXR_EyebrowFeature_Middle	0	眉毛中心点
GSXR_EyebrowFeature_Left	1	眉毛最左侧点
GSXR_EyebrowFeature_Right	2	眉毛最右侧点
GSXR_EyebrowFeature_Middle_Left	3	眉毛中心点与左侧点的中点
GSXR_EyebrowFeature_Middle_Right	4	眉毛中心点与右侧点的中点

2. **眼睛特征点**：由 11 个特征点组成，定义如下。

```
#define GSXR_EYE_FEATURE_COUNT 11
```

[枚举] GSXR_EyeFeature (眼睛特征点)

名称	数值	描述
GSXR_EyeFeature_Pupil	0	眼球瞳孔中心点
GSXR_EyeFeature_Socket_Left	1	眼窝最左侧点
GSXR_EyeFeature_Socket_Right	2	眼窝最右侧点
GSXR_EyeFeature_Socket_Bottom	3	眼窝最下侧点
GSXR_EyeFeature_Socket_Top	4	眼窝最上侧点
GSXR_EyeFeature_Socket_Left_Bottom	5	眼窝左下侧点
GSXR_EyeFeature_Socket_Right_Bottom	6	眼窝右下侧点
GSXR_EyeFeature_Socket_Left_Top	7	眼窝左上侧点

GSXR_EyeFeature_Socket_Right_Top	8	眼窝右上侧点
GSXR_EyeFeature_Iris_Left	9	眼球虹膜左侧点
GSXR_EyeFeature_Iris_Right	10	眼球虹膜右侧点

3. **鼻子特征点：**由 9 个特征点组成，定义如下。

```
#define GSXR_NOSE_FEATURE_COUNT 9
```

[枚举] GSXR_NoseFeature（鼻子特征点）

名称	数值	描述
GSXR_NoseFeature_Tip	0	鼻尖点
GSXR_NoseFeature_Bottom	1	鼻子最下侧点
GSXR_NoseFeature_Bridge	2	鼻梁最上侧点
GSXR_NoseFeature_Wing_Left	3	左侧鼻翼点
GSXR_NoseFeature_Wing_Right	4	右侧鼻翼点
GSXR_NoseFeature_Wing_Left_Outer	5	左侧鼻翼最外侧点
GSXR_NoseFeature_Wing_Right_Outer	6	右侧鼻翼最外侧点
GSXR_NoseFeature_Wing_Left_Lower	7	左侧鼻翼最下侧点
GSXR_NoseFeature_Wing_Right_Lower	8	右侧鼻翼最下侧点

4. **嘴巴特征点：**由 14 个特征点组成，定义如下。

```
#define GSXR_MOUTH_FEATURE_COUNT 14
```

[枚举] GSXR_MouthFeature（嘴巴特征点）

名称	数值	描述
GSXR_MouthFeature_Left	0	嘴巴最左侧点
GSXR_MouthFeature_Right	1	嘴巴最右侧点
GSXR_MouthFeature_Lip_Bottom	2	下嘴唇最下侧点
GSXR_MouthFeature_Lip_Bottom_Inner	3	下嘴唇最下侧的内侧点
GSXR_MouthFeature_Lip_Top	4	上嘴唇最上侧点
GSXR_MouthFeature_Lip_Top_Inner	5	上嘴唇最上侧的内侧点
GSXR_MouthFeature_Lip_Left_Bottom	6	下嘴唇左下侧点
GSXR_MouthFeature_Lip_Left_Bottom_Inner	7	下嘴唇左下侧的内侧点
GSXR_MouthFeature_Lip_Right_Bottom	8	下嘴唇右下侧点
GSXR_MouthFeature_Lip_Right_Bottom_Inner	9	下嘴唇右下侧的内侧点
GSXR_MouthFeature_Lip_Left_Top	10	上嘴唇左上侧点
GSXR_MouthFeature_Lip_Left_Top_Inner	11	上嘴唇左上侧的内侧点
GSXR_MouthFeature_Lip_Right_Top	12	上嘴唇右上侧点
GSXR_MouthFeature_Lip_Right_Top_Inner	13	上嘴唇右上侧的内侧点

5. **脸廓特征点**：由 9 个特征点组成，定义如下。

```
#define GSXR_CONTOUR_FEATURE_COUNT 9
```

[枚举] GSXR_ContourFeature（脸廓特征点）

名称	数值	描述
GSXR_ContourFeature_Jawline_Left_Upper	0	颚线左方上侧点
GSXR_ContourFeature_Jawline_Right_Upper	1	颚线右方上侧点
GSXR_ContourFeature_Jawline_Left_Middle	2	颚线左方中点

GSXR_ContourFeature_Jawline_Right_Middle	3	颚线右方中点
GSXR_ContourFeature_Jawline_Left_Lower	4	颚线左方下侧点
GSXR_ContourFeature_Jawline_Right_Lower	5	颚线右方下侧点
GSXR_ContourFeature_Chin_Left	6	下巴左侧点
GSXR_ContourFeature_Chin_Right	7	下巴右侧点
GSXR_ContourFeature_Chin_Bottom	8	下巴最下侧点

除上述脸部部位的详细特征点外，也可以是脸部主体提取的 5 个脸部主特征点。

```
#define GSXR_FACIAL_MAIN_FEATURE_COUNT 5
```

[枚举] GSXR_FacialMainFeature (脸部主特征点)

名称	数值	描述
GSXR_FacialMainFeature_Eye_Left	0	左眼球中心点
GSXR_FacialMainFeature_Eye_Right	1	右眼球中心点
GSXR_FacialMainFeature_Nose	2	鼻尖点
GSXR_FacialMainFeature_Mouth_Left	3	嘴巴左侧点
GSXR_FacialMainFeature_Mouth_Right	4	嘴巴右侧点

脸部部位的特征点数据由 GSXR_FacialPortionData 结构体描述，结构中包含了该部位的特征点数目、各特征点的有效性、以及各特征点坐标，特征点坐标为目标图像的(x, y)坐标。

[结构体] GSXR_FacialPortionData (脸部部位特征点数据)

```
typedef struct GSXR_FacialPortionData {
   void*                next;
   uint32_t             featureCount;
   GSXR_Bool32*         isFeatureValid;
   GSXR_FacialPoint*    features;
} GSXR_FacialPortionData;
```

说明：

脸部部位特征点数据结构体

成员：

next 预留指针

featureCount 部位特征点数目

isFeatureValid 部位特征点有效性数组

features 部位特征点坐标数组

[结构体] GSXR_FacialPoint (脸部特征点坐标)

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

说明：

脸部特征点坐标结构体

成员：

x 特征点 x 坐标

y 特征点 y 坐标

脸部主特征点的位置数据由 GSXR_FacialMainFeatureData 结构体描述，XR 应用程序可调用 GSXR_GetFacialMainFeatureData 函数获取。脸部详细特征点数据则描述于 GSXR_FacialDetailFeatureData 结构体，若 XR 应用需求更详细的特征点数据，可调用 GSXR_GetFacialDetailFeatureData 函数获取。

[结构体] GSXR_FacialMainFeatureData (脸部主特征点数据)

```
typedef struct GSXR_FacialMainFeatureData {
    void*                       next;
    GSXR_FacialPortionData      main;
} GSXR_FacialMainFeatureData;
```

说明：

脸部主特征点数据结构体

成员：

next 预留指针

main 脸部主特征点数据，featureCount 为 GSXR_FACIAL_MAIN_FEATURE_COUNT

[结构体] GSXR_FacialDetailFeatureData (脸部详细特征点数据)

```
typedef struct GSXR_FacialDetailFeatureData {
    void*                       next;
    GSXR_FacialPortionData      leftEyebrow;
    GSXR_FacialPortionData      rightEyebrow;
    GSXR_FacialPortionData      leftEye;
    GSXR_FacialPortionData      rightEye;
    GSXR_FacialPortionData      nose;
    GSXR_FacialPortionData      mouth;
    GSXR_FacialPortionData      contour;
} GSXR_FacialDetailFeatureData;
```

说明：

脸部详细特征点数据结构体

成员：

next 预留指针

leftEyebrow 左眉毛特征点数据，featureCount 为 GSXR_EYEBROW_FEATURE_COUNT

rightEyebrow 右眉毛特征点数据，featureCount 为 GSXR_EYEBROW_FEATURE_COUNT

leftEye 左眼特征点数据，featureCount 为 GSXR_EYE_FEATURE_COUNT

rightEye 右眼特征点数据，featureCount 为 GSXR_EYE_FEATURE_COUNT

nose 鼻子特征点数据，featureCount 为 GSXR_NOSE_FEATURE_COUNT

mouth 嘴巴特征点数据，featureCount 为 GSXR_MOUTH_FEATURE_COUNT

contour 脸廓特征点数据，featureCount 为 GSXR_CONTOUR_FEATURE_COUNT

[函数] GSXR_GetFacialMainFeatureData（获取脸部主特征点数据）

```
GSXR_Result GSXR_GetFacialMainFeatureData (
    GSXR_XrFeature                  feature,
    GSXR_FaceId                     faceId,
    GSXR_FacialMainFeatureData*     data);
```

说明：

获取脸部主特征点数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] faceId 指定脸部标识号

[out] data 指向 GSXR_FacialMainFeatureData 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

GSXR_FacialPortionData 中的特征点数组内存由应用程序根据特征点数目配置

[基础能力规范] GSXR_GetFacialMainFeatureData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `faceId` 必须是有效的脸部标识号。
- `data` 必须是指向 GSXR_FacialMainFeatureData 结构的有效指针。

[函数实现]

- 脸部识别特性将 faceId 图像的脸部主特征点信息填入 data 中输出。

[返回值]

成功

- GSXR_Result_Success 脸部主特征点数据获取成功。

失败

- GSXR_Error_Operation_Failed 脸部主特征点数据获取失败。
- GSXR_Error_Feature_Unsupported 不支持脸部识别特性。
- GSXR_Error_Handle_Invalid feature 为无效句柄。
- GSXR_Error_Argument_Invalid data 为 nullptr。或 faceId 为一无效的脸部标识号。
- GSXR_Error_Runtime_Unavailable Runtime 未处于正确工作状态。
- GSXR_Error_Feature_Unavailable 特性未处于正确工作状态。

[函数] GSXR_GetFacialDetailFeatureData（获取脸部详细特征点数据）

```
GSXR_Result GSXR_GetFacialDetailFeatureData (
      GSXR_XrFeature                    feature,
      GSXR_FaceId                       faceId,
      GSXR_FacialDetailFeatureData*     data);
```

说明：

获取脸部详细特征点数据

参数：

[in] feature GSXR_XrFeature 句柄，由 GSXR_StartXrFeature 获取

[in] faceId 指定脸部标识号

[out] data 指向 GSXR_FacialDetailFeatureData 结构体

返回值：

GSXR_Result_Success

GSXR_Error_Operation_Failed

GSXR_Error_Feature_Unsupported

GSXR_Error_Handle_Invalid

GSXR_Error_Argument_Invalid

GSXR_Error_Runtime_Unavailable

GSXR_Error_Feature_Unavailable

备注：

GSXR_FacialPortionData 中的特征点数组内存由应用程序根据特征点数目配置

[基础能力规范] GSXR_GetFacialDetailFeatureData

[参数检验]

- `feature` 必须是有效的 GSXR_XrFeature 句柄。
- `faceId` 必须是有效的脸部标识号。
- `data` 必须是指向 GSXR_FacialDetailFeatureData 结构的有效指针。

[函数实现]

- 脸部识别特性将 faceId 图像的脸部详细特征点信息填入 data 中输出。

[返回值]

成功

- `GSXR_Result_Success` 脸部详细特征点数据获取成功。

失败

- `GSXR_Error_Operation_Failed` 脸部详细特征点数据获取失败。
- `GSXR_Error_Feature_Unsupported` 不支持脸部识别特性。
- `GSXR_Error_Handle_Invalid` feature 为无效句柄。
- `GSXR_Error_Argument_Invalid` data 为 nullptr。或 faceId 为一无效的脸部标识号。
- `GSXR_Error_Runtime_Unavailable` Runtime 未处于正确工作状态。
- `GSXR_Error_Feature_Unavailable` 特性未处于正确工作状态。

12. 附录

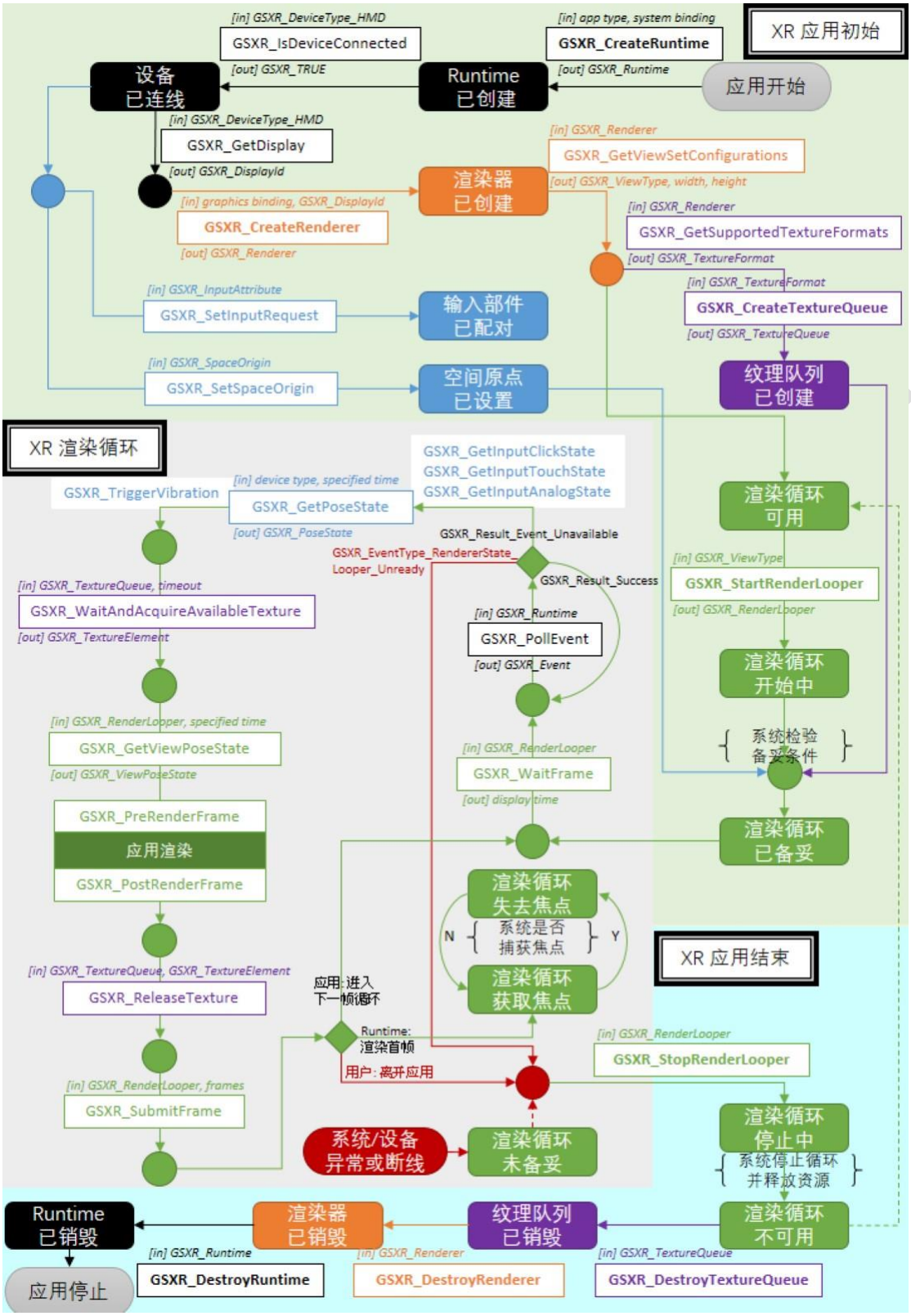
12.1. 附录一 GSXR 互通接口适用对象

本规范适用于以下对象，各对象关注章节如下表。

适用对象	关注章节
应用开发者	所有章节（第 10 章为选用章节，应用开发者可依据内容特性选择对应章节阅读）
Runtime 厂家	所有章节（第 10 章为选用章节，Runtime 厂家可依据自身支持的特性选择对应章节阅读）
头显设备厂家	7 XR 设备函数说明
手柄设备厂家	7 XR 设备函数说明 10.1 XR 特性通用函数 10.5 XR 特性 - 手柄模型
眼球追踪算法厂家	7.6 空间位姿 10.1 XR 特性通用函数 10.2 XR 特性 - 眼球追踪
手部追踪算法厂家	7.6 空间位姿 10.1 XR 特性通用函数 10.3 XR 特性 - 手部追踪
语音命令算法厂家	10.1 XR 特性通用函数， 10.4 XR 特性 - 语音命令
点云数据算法厂家	10.1 XR 特性通用函数， 10.6 XR 特性 - 点云数据
平面侦测算法厂家	10.1 XR 特性通用函数，

	10.7 XR 特性 - 平面侦测
图像识别算法厂家	10.1 XR 特性通用函数, 10.8 XR 特性 - 图像识别
脸部识别算法厂家	10.1 XR 特性通用函数, 10.9 XR 特性 - 脸部识别

12.2. 附录二　GSXR 互通接口主线流程



12.3. 附录三　GSXR 互通接口函数索引

12.3.1. XR Runtime 函数

- GSXR_ResultToString(返回值转换字符串)
- GSXR_CreateRuntime(创建 Runtime)
- GSXR_DestroyRuntime(销毁 Runtime)
- GSXR_GetRuntimeProperties(获取 Runtime 属性)
- GSXR_PollEvent(轮询事件队列获取事件)

12.3.2. XR 设备函数

- GSXR_GetSupportedDevices(获取支持的设备类型)
- GSXR_GetSupportedDeviceInputType (获取支持的控制器类型)
- GSXR_IsDeviceConnected(获取设备连线状态)
- GSXR_GetDisplay(获取显示设备标识号)
- GSXR_GetInputIdConfiguration(获取输入部件具备的输入类型配置)
- GSXR_GetInputClickStates(获取设备输入部件点击状态)
- GSXR_GetInputTouchStates(获取设备输入部件触摸状态)
- GSXR_GetInputAnalogState(获取设备输入部件类比数据)
- GSXR_TriggerVibration(触发设备振动反馈)
- GSXR_GetDegreeOfFreedom(获取设备空间自由度跟踪能力)
- GSXR_SetSpaceOrigin(设置 XR 应用的空间原点及偏移量)
- GSXR_GetPoseState(获取设备于指定时间的位姿及速度数据)
- GSXR_GetSpaceBoundsConfiguration(获取空间边界配置信息)
- GSXR_GetSpaceBoundsRectangle(获取矩形边界信息)
- GSXR_GetSpaceBoundsCircle(获取圆形边界信息)
- GSXR_GetSpaceBoundsIrregular(获取不规则形边界信息)
- GSXR_SetSpaceBoundsVisible(设置边界防护墙可见模式)
- GSXR_GetDeviceProperties(获取设备属性)
- GSXR_GetBatteryStatus(获取设备电池状态)

12.3.3. XR DM 上报函数

- GSXR_GetDMData(DM 指定类型数据获取)

- GSXR_SetDmEventCallback(设置 DM 运行时事件回调)
- GSXR_DMReportedInfo(获取设备 DM 数据结构基础信息)

12.3.4. XR 渲染器函数

- GSXR_CreateRenderer(创建渲染器)
- GSXR_DestroyRenderer(销毁渲染器)
- GSXR_GetViewSetCount(获取渲染器的视图集数目)
- GSXR_GetViewSetConfigurations(获取渲染器的视图集配置)
- GSXR_GetTextureQueueOptions(获取纹理队列功能选项)
- GSXR_GetSupportedTextureFormats(获取纹理队列的纹理格式)
- GSXR_CreateTextureQueue(创建纹理队列)
- GSXR_DestroyTextureQueue(销毁纹理队列)
- GSXR_GetTextureCount(获取纹理队列中的纹理数目)
- GSXR_WaitAndAcquireAvailableTexture(等待获得可用纹理)
- GSXR_ReleaseTexture(将纹理释放回纹理队列)
- GSXR_StartRenderLooper(开始渲染循环)
- GSXR_StopRenderLooper(停止渲染循环)
- GSXR_SetRenderLooperThread(设置渲染循环线程 id)
- GSXR_WaitFrame(等待调节帧渲染时机)
- GSXR_GetViewPoseState(获取视图于指定时间的姿态)
- GSXR_SubmitFrame(提交渲染帧)
- GSXR_GetRenderFunctions(获取渲染功能)
- GSXR_PreRenderFrame(预设置渲染帧功能)
- GSXR_PostRenderFrame(后设置渲染帧功能)
- GSXR_GetFoveatedParameters(获取现存注视点参数)

12.3.5. XR 系统函数

- GSXR_SetPerformanceLevels(设置 CPU, GPU 性能等级)

12.3.6. XR 特性函数

- GSXR_GetSupportedXrFeatures(获取支持的特性类型)
- GSXR_IsXrFeatureAvailable(获取特性可用状态)

- GSXR_GetXrFeatureOptions(获取特性的功能选项)
- GSXR_StartXrFeature(起始特性运行)
- GSXR_StopXrFeature(停止特性运行)
- GSXR_GetXrFeatureProperties(获取特性属性)

[眼球追踪]
- GSXR_GetSupportedEyeFactors(获取支持的眼球数据类型)
- GSXR_SetEyeFactorRequest(设置使用的眼球数据类型)
- GSXR_GetEyeTrackingData(获取眼球追踪数据)

[手部追踪]
- GSXR_GetSupportedHandGestures(获取可识别的手势类型)
- GSXR_SetHandGestureRequest(设置使用的手势类型)
- GSXR_GetHandGestureData(获取手势数据)
- GSXR_RegisterHandGestureEventCallback(注册手势识别事件回调函数)
- GSXR_GetTrackedHandJoints(获取可被追踪的手部节点)
- GSXR_GetHandJointValidity(获取手部节点空间追踪能力)
- GSXR_GetHandTrackingData(获取手部追踪数据)

[语音命令]
- GSXR_InputVoiceCommand(输入语音命令)

[手柄模型]
- GSXR_AcquireControllerModelData(获得手柄模型数据)
- GSXR_ReleaseControllerModelData(释放手柄模型数据)

[点云数据]
- GSXR_AcquirePointCloudData(获得点云数据)
- GSXR_ReleasePointCloudData(释放点云数据)

[平面侦测]
- GSXR_AcquirePlaneTrackerData(获得平面数据)
- GSXR_ReleasePlaneTrackerData(释放平面数据)

[图像识别]
- GSXR_LoadModelImage(加载欲识别的模型图像)
- GSXR_UnloadModelImage(卸载模型图像)
- GSXR_AcquireImageTrackerData(获得图像识别结果数据)

- GSXR_ReleaseImageTrackerData(释放图像识别结果数据)

[脸部识别]

- GSXR_AcquireFaceTrackerData(获得脸部识别结果数据)
- GSXR_ReleaseFaceTrackerData(释放脸部识别结果数据)
- GSXR_AcquireFaceMatchTrackerData(获得脸部比对结果数据)
- GSXR_ReleaseFaceMatchTrackerData(释放脸部比对结果数据)
- GSXR_GetFacialMainFeatureData(获取脸部主特征点数据)
- GSXR_GetFacialDetailFeatureData(获取脸部详细特征点数据)

中国 VRPC 联盟标准

GSXR 互通接口与基础能力规范

GSXR-A-01-04