联　　盟　　标　　准

GSXR 互通设备层接口与基础能力规范
GSXR-D-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)互通设备规范，提供了XR设备标准API（Application Programming Interface）接口定义与功能描述，XR设备接口包括头戴式显示器、手柄控制器、及运行系统等类别。

XR设备厂家可以使用格式一致的XR功能函数开发XR设备插件，无需再为不同Runtime的设备接口进行代码适配，设备厂家可专注于XR设备研发，且开发出的GSXR设备插件可适配各个GSXR Runtime。目前，此规范只适用于安卓系统。

目 录

1. 概述

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

GSXR互通设备接口是一系列为XR设备制定的函数集，依据XR设备类型区分类别，定义各设备函数相应的输入输出参数及返回值。本规范将根据XR互通设备函数，描述详细的接口定义，以及XR Runtime及XR设备插件实际的操作及实现细项。

2. 适用范围

XR设备插件

XR设备厂家根据设备所支持的功能，提供对应的XR函数实现，例如上报头显或手柄设备状态、设备跟踪及输入数据等,设备厂家必须依据自身设备的硬件、固件、及算法设计实现GSXR设备接口，以提供Runtime需求的设备数据。

XR Runtime

XR Runtime必须根据产品设计，正确完成XR设备插件的初始化，并依据XR应用的功能需求,在适当的使用时机,调用对应的GSXR设备接口获取设备数据，配合XR Runtime本身的架构及功能设计，提供XR应用程序即时的设备交互数据，确保XR应用在XR设备上正确运行。

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 互通设备规范
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. XR 设备通用基础类型

本章节描述 GSXR 互通设备接口的基本类型定义，后续章节的 XR 设备功能函数都将以本章为通用原则延伸其功能定义。

4.1. XR 设备接口版本号

版本号 gsxr_version 为 GSXR 互通设备接口描述版本信息的标准类型，XR 设备插件必须根据实现的 GSXR 设备接口版本上报正确的版本号。

```
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设立新接口，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_DEVICE_CURRENT_API_VERSION：获取当前 GSXR 设备接口版本号
  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
```

4.2. 设备句柄

GSXR 互通设备接口创建并开启设备实例时，设备句柄便为此设备实例的代表，XR Runtime 调用设备函数时都需要指定对应句柄为输入参数，用以识别操作的设备实例对象。

```
typedef void* gsxr_dev_handle_t;
```

句柄的生成代表着设备实例的生成，其生命周期由一组成对的函数所管控（开启 *open / 关闭 *close），XR Runtime 必须适时调用设备的开启与关闭函数，正确管控设备实例的生命周期。

4.3. 设备识别号

同一设备类型可能同时存在着多个设备（如：手柄控制器有左右手之分），GSXR 互通设备接口定义了设备识别号识别设备实体对象，XR Runtime 操作设备函数时必须输入对应的设备识别号，XR 设备插件将根据设备识别号操作对应的实体设备。

```
typedef uint32_t gsxr_dev_id_t;
```

设备识别号的生成由 XR 设备插件在设备连接时经由事件回调函数返回给 XR Runtime，有效的标识号不可为 0。

4.4. 设备函数返回值

设备函数返回值返回 0 代表执行成功，返回负值则代表执行失败或遭遇某种非预期的错误状态，各设备插件需根据当前状态返回对应的错误代码。XR Runtime 厂家及设备厂家在产品集成时必须确保在可控的使用行为下，各设备函数的执行全部成功且返回值为 0。

4.5. 设备回调事件类型及函数指针

XR 设备插件根据功能类型，在某些特定状态触发时，必须回调 XR Runtime 所注册的事件函数指针，将当前 XR 设备所发生的状态通知到 XR Runtime。

[类型定义] gsxr_dev_event_callback_fn（设备事件回调函数）

```
typedef void (*gsxr_dev_event_callback_fn) (
      gsxr_dev_event_callback_type_t event,
      void* param1,
      void* param2,
      void* param3,
      void* cookie);
```

说明：

设备事件回调函数，当设备触发新事件时回调

参数：

[in] event 设备回调事件类型

[in] param1 回调参数 1

[in] param2 回调参数 2

[in] param3 回调参数 3

[in] cookie 函数回调时带回的预备数值

返回值：

无

备注：

各回调参数按回调事件类型具有不同定义，请参阅各事件说明进行相应的类型转换，赋予实值

[枚举] gsxr_dev_event_callback_type_t

1. 设备状态事件

名称	数值	描述
GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED	2000	设备连接
GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_DISCONNECTED	2001	设备断开
[回调参数定义] • `param1` 设备识别号(gsxr_dev_id_t)		

名称	数值	描述
GSXR_DEV_EVENT_CALLBACK_TYPE_TRACKING_MODE_CHANGED	2003	设备跟踪模式变换
[回调参数定义] • param1 设备识别号(gsxr_dev_id_t) • param2 新跟踪模式(gsxr_tracking_mode_t) • param3 前跟踪模式(gsxr_tracking_mode_t)		

名称	数值	描述
GSXR_DEV_EVENT_CALLBACK_TYPE_TRACKING_STATE_CHANGED	2499	跟踪系统状态改变
[回调参数定义] • param1 设备识别号(gsxr_dev_id_t) • param2 新跟踪状态(gsxr_tracking_state_t) • param3 前跟踪状态(gsxr_tracking_state_t)		

2. 设备输入事件

名称	数值	描述
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_CLICKED	3000	起始点击
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_UNCLICKED	3001	结束点击
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_TOUCHED	3002	起始触控
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_UNTOUCHED	3003	结束触控
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_LEFT_TO_RIGHT_SWIPED	3004	左至右滑动
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_RIGHT_TO_LEFT_SWIPED	3005	右至左滑动
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_TOP_TO_BOTTOM_SWIPED	3006	上至下滑动
GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_BOTTOM_TO_TOP_SWIPED	3007	下至上滑动
[回调参数定义]		

- `param1` 设备识别号(gsxr_dev_id_t)
- `param2` 输入部件识别号(gsxr_input_id_t)

4.6. 设备坐标系及姿态

GSXR 采用右手坐标系。

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_t (二维矢量)

```
typedef struct gsxr_vector2f {
    float    x;
    float    y;
} gsxr_vector2f_t;
```

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_t (三维矢量)

```
typedef struct gsxr_vector3f {
    float    x;
    float    y;
    float    z;
```

```
} gsxr_vector3f_t;
```

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_t（四元数）

```
typedef struct gsxr_quaternionf {
    float    x;
    float    y;
    float    z;
    float    w;
} gsxr_quaternionf_t;
```

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_t（位姿）

```
typedef struct gsxr_posef {
    gsxr_vector3f_t       position;
    gsxr_quaternionf_t    orientation;
} gsxr_posef_t;
```

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_t (速度)

```
typedef struct gsxr_velocityf {
    gsxr_vector3f_t       linear_velocity;
    gsxr_vector3f_t       angular_velocity;
} gsxr_velocityf_t;
```

说明：

速度结构体，用以表示线速度及角速度

成员：

linear_velocity 表示线速度

angular_velocity 表示角速度

5. XR 设备通用函数说明

对于各种不同类型的 XR 设备可经由同一接口形式实现的函数，GSXR 通用设备接口将其归类于通用函数结构体 gsxr_dev_common_ops_t 之中。各 XR 设备插件(头显、手柄、系统)的实现函数全部包含此通用函数结构，设备插件可按需求实现对应的通用函数。

```
typedef struct gsxr_dev_common_ops {
    gsxr_dev_handle_t (*open)(void);
    void (*close)(gsxr_dev_handle_t dev_handle);
```

```
    int32_t (*set_event_callback)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_event_callback_fn callback,
                void* cookie);
    int32_t (*get_param)(
                gsxr_dev_handle_t dev_handle,
                const char* name,
                char* value,
                uint32_t* length);
    int32_t (*set_param)(
                gsxr_dev_handle_t dev_handle,
                const char* name,
                const char* value);
} gsxr_dev_common_ops_t;
```

5.1. 开启与关闭设备实例

XR 设备插件必须实现开启(*open)与关闭(*close)函数，此二函数管控 XR 设备实例生命周期的起始与结束，XR Runtime 对于 XR 设备的操作行为必须全部介于此二函数的调用区间。

设备开启函数(*open)若执行成功，XR 设备插件必须返回一个有效的设备句柄 gsxr_dev_handle_t。XR Runtime 对于此设备的所有函数操作都必须输入此设备句柄以做识别，XR Runtime 调用设备函数若输入无效的设备句柄， XR 设备插件将视其为无效的操作。当 XR Runtime 不再使用此设备时，必须调用关闭函数(*close)完成设备资源的释放。

[函数] gsxr_dev_common_ops_t (*open) (开启设备实例)

```
gsxr_dev_handle_t (*open)(void);
```

说明：

开启设备实例，获取设备句柄

参数：

无

返回值：

设备句柄 gsxr_dev_handle_t

备注：

无

[基础能力规范] gsxr_dev_common_ops_t (*open)

[参数检验]

无

[函数实现]

- XR 设备插件正确初始设备实例，并返回有效的设备句柄。

[返回值]

成功

- 返回有效的设备句柄，数值大于 0。

失败

- 返回无效的设备句柄 0。

[函数] gsxr_dev_common_ops_t (*close) (关闭设备实例)

```
void (*close)(gsxr_dev_handle_t dev_handle);
```

说明：

关闭设备实例

参数：

[in] dev_handle 设备句柄，由 open 函数获取

返回值：

无

备注：

无

[基础能力规范] gsxr_dev_common_ops_t (*close)

[参数检验]

- dev_handle 必须是有效的设备句柄。

[函数实现]

- XR 设备插件正确释放设备实例资源。

[返回值]

无

5.2. 设置事件回调函数

XR Runtime 必须设置事件回调函数指针给 XR 设备插件，XR 设备插件在指定的设备事件发生时必须回调此函数指针执行 XR Runtime 对应的事件处理行为，XR Runtime 设计回调函数必须考虑回调线程的即时性，不可用于回调函数中执行过于耗时的 Runtime 工作。

[函数] gsxr_dev_common_ops_t (*set_event_callback) (设置事件回调函数)

```
int32_t (*set_event_callback)(
    gsxr_dev_handle_t                dev_handle,
    gsxr_dev_event_callback_fn       callback,
    void*                            cookie);
```

说明：

设置事件回调函数，各设备于指定事件发生时回调

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] callback 事件回调函数指针

[in] cookie 函数回调时带回的预备数值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的回调函数

备注：

无

[基础能力规范] gsxr_dev_common_ops_t (*set_event_callback)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- callback 必须是 gsxr_dev_event_callback_fn 函数类型的有效指针。

[函数实现]

- XR 设备插件记录 XR Runtime 设置的 gsxr_dev_event_callback_fn 函数指针，并根据设备状态于设备事件发生时正确回调函数，设备事件类型请参阅 4.5。

[返回值]

成功

- 0 回调函数设置成功。

失败

- -1 回调函数设置失败。
- -2 dev_handle 为无效句柄。
- -3 callback 为 nullptr。

[基础能力规范] 设备连线事件回调

[功能实现]

- GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED 当设备连接,XR 设备插件完成设备初始化后调用事件回调函数通知 XR Runtime，param1 为设备识别号(gsxr_dev_id_t)。
- GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_DISCONNECTED 当设备断开，XR 设备插件完成资源释放后调用事件回调函数通知 XR Runtime，param1 为设备识别号(gsxr_dev_id_t)。

5.3. 获取与设置设备扩展参数值

考虑 XR 设备可能具有特殊的扩展功能，GSXR 定义了通用的扩展参数获取与设置函数，供 XR Runtime 与 XR 设备插件于特殊扩展功能交互沟通使用，扩展函数通常运用于 XR 设备特定的使用情境。

[函数] gsxr_dev_common_ops_t (*get_param) (获取设备扩展功能参数值)

```
int32_t (*get_param)(
    gsxr_dev_handle_t               dev_handle,
    const char*                     name,
    char*                           value,
    uint32_t*                       length);
```

说明：

获取设备扩展功能参数值，设备有特殊扩展功能时使用

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] name 扩展参数名称，以 '\0' 结尾，不可为 nullptr

[in,out] value value 不为 nullptr 时，输出对应 name 的扩展参数值，以 '\0' 结尾

[in,out] length 当 value 为 nullptr 时，length 输出 value 的字符串长度(包含 '\0')；当 value 不为 nullptr 时，lenth 输入 value 的字符串长度(包含 '\0')；lenth 不可为 nullptr

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_common_ops_t (*get_param)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- name 必须是有效的字符串指针，不可为 nullptr。
- value 当 value 不为 nullptr 时，其字符串长度必须等于 length(包含'\0')。
- length 必须是指向 uint32_t 数值的有效指针，不可为 nullptr。

[函数实现]

- XR 设备插件根据扩展参数名称 name，输出其对应的扩展参数数值字符串 value。
- 当 value 为 nullptr 时，length 输出 value 的字符串长度(包含 '\0')。
- 当 value 不为 nullptr 时，lenth 输入 value 的缓存长度(包含 '\0')，value 输出对应 name 的扩展参数值字符串，以 '\0' 结尾。

[返回值]

成功

- 0 扩展参数获取成功。

失败

- -1 扩展参数获取失败。
- -2 dev_handle 为无效句柄。
- -3 name 为 nullptr 或无效的参数名称。length 为 nullptr。

[函数] gsxr_dev_common_ops_t (*set_param) (设置设备扩展功能参数值)

```
int32_t (*set_param)(
    gsxr_dev_handle_t          dev_handle,
    const char*                name,
    const char*                value);
```

说明：

设置设备扩展功能参数值，设备有特殊扩展功能时使用

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] name 扩展参数名称，以 '\0' 结尾，不可为 nullptr

[in] value 对应 name 的扩展参数值字符串，以 '\0' 结尾，不可为 nullptr

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_common_ops_t (*set_param)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- name 必须是有效的字符串指针，不可为 nullptr。
- value 必须是有效的字符串指针，不可为 nullptr。

[函数实现]

- XR 设备插件根据扩展参数名称 name，设置其对应的扩展参数数值字符串，并执行设备特殊定义的对应行为(若有)。

[返回值]

成功

- 0 扩展参数设置成功。

失败

- -1 扩展参数设置失败。
- -2 dev_handle 为无效句柄。
- -3 name 为 nullptr 或无效的参数名称。或 value 为 nullptr。

6. XR 设备信息函数说明

GSXR 通用设备接口将各种 XR 设备的信息状态函数归类于信息函数结构体 gsxr_dev_info_ops_t 之中。

```
typedef struct gsxr_dev_info_ops {
   int32_t (*get_dev_properties)(gsxr_dev_handle_t dev_handle,
                                                    gsxr_dev_id_t dev_id,
                                                    gsxr_dev_properties_t*
                                                    properties);
   int32_t (*get_battery_status)(gsxr_dev_handle_t dev_handle,
                                                    gsxr_dev_id_t dev_id,
                                                    gsxr_battery_status_t*
                                                    battery_status);

   int32_t (*set_dm_event_callback)(gsxr_dev_handle_t dev_handle,
                                                    gsxr_dm_event_callback_fn
                                                    callback);

   int32_t (*get_dm_reported_info)(gsxr_dev_handle_t dev_handle,
                                                     gsxr_dm_reported_info_t*
                                                    dm_reported_info);

   int32_t (*get_dm_data)(gsxr_dev_handle_t dev_handle,
                                                    gsxr_dev_id_t dev_id,

                                                    gsxr_timing_gather_t
                                                    timing_gather_Type,

                                                    gsxr_timing_gather_info_t*
                                                    timing_gather);

} gsxr_dev_info_ops_t;
```

6.1. 获取设备属性

XR 设备属性定义在结构体 gsxr_dev_properties_t 中，可以由函数

get_dev_properties 获取。

[结构体] gsxr_dev_properties_t (设备属性)

```
typedef struct gsxr_dev_properties {
    char            serial_number[GSXR_COMMON_STRING_MAX_SIZE];
    char            firmware_version[GSXR_COMMON_STRING_MAX_SIZE];
    char            vendor_name[GSXR_COMMON_STRING_MAX_SIZE];
    char            algorithm_version[GSXR_COMMON_STRING_MAX_SIZE];
    char            algorithm_name[GSXR_COMMON_STRING_MAX_SIZE];
} gsxr_dev_properties_t;
```

说明：

设备属性结构体

成员：

serial_number 设备 SN 串号

firmware_version 设备 firmware 版本号

vendor_name 设备厂家名称

algorithm_version 设备算法版本号

algorithm_name 设备算法名称

[函数] gsxr_dev_info_ops_t (*get_dev_properties) (获取设备属性)

```
int32_t (*get_dev_properties)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    gsxr_dev_properties_t*          properties);
```

说明：

获取设备属性

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] properties 指向 gsxr_dev_properties_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_info_ops_t (*get_dev_properties)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `properties` 必须是指向 gsxr_dev_properties_t 结构的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的设备属性，填入 properties 中输出。

[返回值]

成功

- `0` 设备属性获取成功。

失败

- `-1` 设备属性获取失败。
- `-2` dev_handle 为无效句柄。
- `-3` properties 为 nullptr。
- `-4` dev_id 为无效设备识别号。

6.2. 获取设备电池状态

XR 设备电池状态定义于结构体 gsxr_battery_status_t 中，可经由函数

get_battery_status 获取。

[结构体] gsxr_battery_status_t (设备电池状态)

```
typedef struct gsxr_battery_status {
    float           percentange;
    bool            charging;
} gsxr_battery_status_t;
```

说明：

设备电池状态结构体

成员：

percentange 电池剩余电量百分比, 0 - 100

charging true 表示电池处于充电状态, false 表示电池未处于充电状态

[函数] gsxr_dev_info_ops_t (*get_battery_status) (获取设备电量状态)

```
int32_t (*get_battery_status)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    gsxr_battery_status_t*          battery_status);
```

说明：

获取设备电量状态

参数：

[in] dev_handle 设备句柄, 由 open 函数获取

[in] dev_id 设备识别号, 当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] battery_status 指向 gsxr_battery_status_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_info_ops_t (*get_battery_status)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。
- battery_status 必须是指向 gsxr_battery_status_t 结构的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的电池状态，填入 battery_status 中输出。

[返回值]

成功

- 0 设备电池状态获取成功。

失败

- -1 设备电池状态获取失败。
- -2 dev_handle 为无效句柄。
- -3 battery_status 为 nullptr。
- -4 dev_id 为无效设备识别号。

7. XR 设备输入函数说明

GSXR 通用设备接口将 XR 设备的输入数据函数归类于输入函数结构体 gsxr_dev_input_ops_t 之中。

```
typedef struct gsxr_dev_input_ops {
```

```
    int32_t(*get_supported_device_inputtype)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id,
                uint32_t* device_inputtypeflag);
    int32_t (*get_input_configurations)(
                    gsxr_dev_handle_t dev_handle,
                    gsxr_dev_id_t dev_id,
                    gsxr_device_inputtype dev_inputtype,
                    gsxr_dev_input_configurations_t* input_configs);
    int32_t (*get_input_click_states)(
                    gsxr_dev_handle_t dev_handle,
                    gsxr_dev_id_t dev_id,
                    gsxr_device_inputtype dev_inputtype,
                    uint64_t* buttons);
    int32_t (*get_input_touch_states)(
                    gsxr_dev_handle_t dev_handle,
                    gsxr_dev_id_t dev_id,
                    gsxr_device_inputtype dev_inputtype,
                    uint64_t* touchs);
    int32_t (*get_input_analog_state)(
                    gsxr_dev_handle_t dev_handle,
                    gsxr_dev_id_t dev_id,
                    gsxr_device_inputtype dev_inputtype,
                    int input_id,
                    gsxr_vector2f_t* analog);
} gsxr_dev_input_ops_t;
```

GSXR 定义以下输入部件及输入类型，XR 设备必须为其所支持的部件提供即时的输入数据供 XR Runtime 获取。

[枚举] gsxr_joystick_inputid_t (设备输入类型)

名称	数值	描述
GSXR_JOYSTICK_INPUTID_A	1	A 键

GSXR_JOYSTICK_INPUTID_B	2	B 键
GSXR_JOYSTICK_INPUTID_X	3	X 键
GSXR_JOYSTICK_INPUTID_Y	4	Y 键
GSXR_JOYSTICK_INPUTID_SYSTEM	5	系统键
GSXR_JOYSTICK_INPUTID_MENU	6	菜单键
GSXR_JOYSTICK_INPUTID_GRIP	7	抓握键
GSXR_JOYSTICK_INPUTID_TRIGGER	8	扳机键
GSXR_JOYSTICK_INPUTID_THUMBSTICK	9	摇杆键
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_touch_inputid_t (设备输入类型)

名称	数值	描述
GSXR_TOUCH_INPUTID_SYSTEM	1	系统键
GSXR_TOUCH_INPUTID_MENU	2	菜单键
GSXR_TOUCH_INPUTID_GRIP	3	抓握键
GSXR_TOUCH_INPUTID_TRIGGER	4	扳机键
GSXR_TOUCH_INPUTID_TRACKPAD	5	触摸板键
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_gamepad_inputid_t (输入部件)

名称	数值	描述
GSXR_GAMEPAD_INPUTID_A	1	A 键
GSXR_GAMEPAD_INPUTID_B	2	B 键
GSXR_GAMEPAD_INPUTID_X	3	X 键
GSXR_GAMEPAD_INPUTID_Y	4	Y 键
GSXR_GAMEPAD_INPUTID_LEFT_THUMBSTICK	5	摇杆键
GSXR_GAMEPAD_INPUTID_RIGHT_THUMBSTICK	6	摇杆键
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	扳机键
GSXR_GAMEPAD_INPUTID_LEFT_SHOULDER	20	shoulder 键
GSXR_GAMEPAD_INPUTID_RIGHT_TRIGGER	21	扳机键
GSXR_GAMEPAD_INPUTID_RIGHT_SHOULDER	22	shoulder 键

[枚举] gsxr_device_inputtype_t（设备输入类型）

名称	数值	描述
GSXR_NO_SUPPORTED_INPUTTYPE	0x01	没有支持的输入类型
GSXR_TOUCH	0x02	触摸版类型控制器，如 HTC 控制器
GSXR_JOYSTICK	0x04	摇杆类型控制器，如 Oculus Quest 控制器
GSXR_GAMEAPD	0x08	Gamepad 输入类型

[枚举] gsxr_inputid_input_type_t（输入类型）

名称	数值	描述
GSXR_INPUT_TYPE_CLICK	0x01	点击类
GSXR_INPUT_TYPE_TOUCH	0x02	触摸类
GSXR_INPUT_TYPE_ANALOG_1D	0x04	类比类 - 一维数据
GSXR_INPUT_TYPE_ANALOG_2D	0x08	类比类 - 二维数据

7.1. 获取输入部件配置

GSXR 定义以下输入部件信息与输入部件配置结构体，并可经由输入部件配置函数获取配置结果。XR 设备插件必须根据设备所支持的输入部件，输出正确的输入部件配置供 XR Runtime 获取。

[结构体] gsxr_dev_input_info_t（输入部件信息）

```
typedef struct gsxr_dev_input_info {
```

```
    int                     input_id;
    uint8_t                 input_type;
} gsxr_dev_input_info_t;
```

说明：

输入部件信息结构体

成员：

input_id 输入部件

input_type 为 gsxr_inputid_input_type_t 类型的位掩码

[结构体] gsxr_dev_input_configurations_t (输入部件配置)

```
typedef struct gsxr_dev_input_configurations {
    uint32_t                            input_len;
    const gsxr_dev_input_info_t*        input_lists;
} gsxr_dev_input_configurations_t;
```

说明：

输入部件配置结构体

成员：

input_len 输入部件数目

input_lists 输入部件信息

[函数] gsxr_dev_input_ops_t (*get_supported_device_inputtype) (获取输入类型配置)

```
int32_t (*get_input_configurations)(
     gsxr_dev_handle_t                      dev_handle,
     gsxr_dev_id_t                          dev_id,
     uint32_t*                              device_inputtypeflag);
```

说明：

获取设备输入类型配置

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] device_inputtypeflag 设备输入类型，为 gsxr_device_inputtype_t 中类型之一

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范]

gsxr_dev_input_ops_t (*get_supported_device_inputtype)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。
- device_inputtypeflag 设备输入类型，为 gsxr_device_inputtype_t。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的输入部件配置，填入 input_configs 中输出。

[返回值]

成功

- 0 设备输入部件配置获取成功。

失败

- -1 设备输入部件配置获取失败。
- -2 dev_handle 为无效句柄。
- -3 input_configs 为 nullptr。

- -4 dev_id 为无效设备识别号。

[函数] gsxr_dev_input_ops_t (*get_input_configurations) (获取输入配置)

```
int32_t (*get_input_configurations)(
    gsxr_dev_handle_t                   dev_handle,
    gsxr_dev_id_t                       dev_id,
    gsxr_device_inputtype_t             dev_inputtype,
    gsxr_dev_input_configurations_t*    input_configs);
```

说明：

获取设备输入部件的配置

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in]dev_inputtype 设备输入类型，由 get_supported_device_inputtype 获取

[out] input_configs 指向 gsxr_dev_input_configurations_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_input_ops_t (*get_input_configurations)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。

- dev_inputtype 必须是有效的设备输入类型
- input_configs 必须是指向 gsxr_dev_input_configurations_t 结构的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的输入部件配置，填入 input_configs 中输出。

[返回值]

成功

- 0 设备输入部件配置获取成功。

失败

- -1 设备输入部件配置获取失败。
- -2 dev_handle 为无效句柄。
- -3 input_configs 为 nullptr。
- -4 dev_id 为无效设备识别号。

7.2. 获取输入状态

GSXR 提供三类函数获取点击状态、触摸状态、及类比数据。

- 调用 get_input_click_states 获取输入类型下所有部件的点击状态。
- 调用 get_input_touch_states 获取输入类型下所有部件的触摸状态。
- 调用 get_input_analog_state 获取指定输入部件的类比数据。

[基础能力规范] 设备输入事件回调

[功能实现]

- 设备输入部件发生输入事件时，XR 设备插件调用事件回调函数通知 XR Runtime，param1 为设备识别号(gsxr_dev_id_t)，param2 为输入部件识别号(gsxr_input_id_t)。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_CLICKED 起始点击。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_UNCLICKED 结束点击。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_TOUCHED 起始触控。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_UNTOUCHED 结束触控。

7.2.1. 点击状态

[函数] gsxr_dev_input_ops_t (*get_input_click_states) (获取点击状态)

```
int32_t (*get_input_click_states)(
    gsxr_dev_handle_t                dev_handle,
    gsxr_dev_id_t                    dev_id,
    gsxr_device_inputtype_t          dev_inputtype,,
    uint64_t*                        clicks);
```

说明：

获取设备输入部件的点击状态

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] dev_inputtype 设备输入类型，由 get_supported_device_inputtype 获取

[out] clicks 按输入类型下的 inputid 为掩码存储每个 inputid 的点击状态信息

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

-5 无效的设备输入类型

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_input_ops_t (*get_input_click_states)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。
- dev_inputtype 必须是有效的设备输入类型。
- clicks 必须是指向 uint64_t 数值的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备 dev_inputtype 输入类型下的所有 inputid 的点击状态填入 clicks 中输出,按 inputid 位掩码存储每个 inputid 点击状态信息。

[返回值]

成功

- 0 设备输入部件点击状态获取成功。

失败

- -1 设备输入部件点击状态获取失败。
- -2 dev_handle 为无效句柄。
- -3 buttons 为 nullptr。
- -4 dev_id 为无效设备识别号。
- -5 dev_inputtype 为无效的输入类型。

7.2.2. 触摸状态

[函数] gsxr_dev_input_ops_t (*get_input_touch_states) (获取触摸状态)

```
int32_t (*get_input_touch_states)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    gsxr_device_inputtype_t         dev_inputtype,
    uint64_t*                       touchs);
```

说明：

获取设备输入部件的触摸状态

参数：

[in] dev_handle 设备句柄, 由 open 函数获取

[in] dev_id 设备识别号, 当设备连接时由回调函数 gsxr_dev_event_callback_fn

获取

[in] dev_inputtype 设备输入类型，由 get_supported_device_inputtype 获取

[out] touchs 按输入类型下的 inputid 为掩码存储每个 inputid 的触摸状态信息

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

-5 无效的设备输入类型

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_input_ops_t (*get_input_touch_state)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `dev_inputtype` 必须是有效的设备输入类型。
- `touchs` 必须是指向 uint64_t 数值的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备 dev_inputtype 输入类型下的所有 inputid 的触摸状态填入 touchs 中输出,按 inputid 位掩码存储每个 inputid 的触摸状态信息。

[返回值]

成功

- 0 设备输入部件触摸状态获取成功。

失败

- -1 备输入部件触摸状态获取失败。
- -2 dev_handle 为无效句柄。
- -3 touchs 为 nullptr。
- -4 dev_id 为无效设备识别号。

- -5 dev_inputtype 为无效的输入类型。

7.2.3. 类比数据

[函数] gsxr_dev_input_ops_t (*get_input_analog_state) (获取类比数据)

```
int32_t (*get_input_analog_state)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    gsxr_device_inputtype_t         dev_inputtype
    int                             input_id,
    gsxr_vector2f_t*                analog);
```

说明：

获取设备输入类型下 inputid 的类比数据

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] dev_inputtype 设备输入类型，由 get_supported_device_inputtype 获取

[in] input_id 指定输入部件，设备输入类型下的 inputid

[out] analog 指向二维矢量 gsxr_vector2f_t 结构体，数据范围见备注说明

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

-5 无效的设备输入部件或者输入类型

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[analog 数据范围]

a. 若输入部件的输入类型为 GSXR_INPUT_TYPE_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_INPUT_TYPE_ANALOG_2D, analog 数据范围为 -1<=x<=1, -1<=y<=1

[基础能力规范] gsxr_dev_input_ops_t (*get_input_analog_state)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。
- dev_inputtype 必须是有效的设备输入类型。
- input_id 必须是有效的输入类型下的 inputid 识别号。
- analog 必须是指向 gsxr_vector2f_t 结构的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备 dev_inputtype 输入类型下的 inputid 输入部件的类比数据填入 analog 中输出。

[返回值]

成功

- 0 设备输入部件类比数据获取成功。

失败

- -1 设备输入部件类比数据获取失败。
- -2 dev_handle 为无效句柄。
- -3 analog 为 nullptr。
- -4 dev_id 为无效设备识别号。
- -5 input_id 为无效的输入部件或者 dev_inputtype 为无效的输入类型。

8. XR 设备跟踪函数说明

GSXR 通用设备接口将 XR 设备的跟踪数据函数归类于跟踪函数结构体 gsxr_dev_tracking_ops_t 之中。

```
typedef struct gsxr_dev_tracking_ops {
    int32_t (*get_tracking_state)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id,
                gsxr_tracking_state_t* tracking_state);
    int32_t (*get_tracking_mode)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id,
                uint32_t* current_mode,
                uint32_t* supported_mode);
    int32_t (*start_tracking)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id,
                uint32_t tracking_mode);
    int32_t (*stop_tracking)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id);
    int32_t (*get_pose_data)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id,
                gsxr_pose_data_t* pose_data);
    int32_t (*get_sensor_data)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id,
                gsxr_sensor_data_t* sensor_data);
    int32_t (*relocalize_origin)(
                gsxr_dev_handle_t dev_handle,
                gsxr_dev_id_t dev_id);
```

```
} gsxr_dev_tracking_ops_t;
```

8.1. 跟踪状态

XR 设备的跟踪系统运行期间，XR Runtime 可经由 get_tracking_state 函数获取当前跟踪系统的工作状态，GSXR 定义以下跟踪系统状态，XR 设备插件必须实时输出当前跟踪系统状态供 XR Runtime 获取。

[枚举] gsxr_tracking_state_t (跟踪系统状态)

名称	数值	描述
GSXR_TRACKING_STATE_UNAVAILABLE	0	跟踪系统不可用
GSXR_TRACKING_STATE_AVAILABLE	1	跟踪系统可用
GSXR_TRACKING_STATE_STARTING	2	跟踪系统起始中
GSXR_TRACKING_STATE_TRACKING	3	跟踪系统工作中
GSXR_TRACKING_STATE_STOPPING	4	跟踪系统停止中
GSXR_TRACKING_STATE_RELOCALIZED	5	跟踪系统重置(*1)
*1 跟踪系统只能在 GSXR_TRACKING_STATE_TRACKING 状态时进行重置， 重置完毕重新进入 GSXR_TRACKING_STATE_TRACKING 状态		

[函数] gsxr_dev_tracking_ops_t (*get_tracking_state) (获取跟踪状态)

```
int32_t (*get_tracking_state)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    gsxr_tracking_state_t*          tracking_state);
```

说明：

获取设备跟踪状态

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] tracking_state 输出当前跟踪状态

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*get_tracking_state)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `tracking_state` 必须是指向 gsxr_tracking_state_t 数值的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的跟踪系统状态填入 tracking_state 中输出。

[返回值]

成功

- `0` 设备跟踪状态获取成功。

失败

- `-1` 设备跟踪状态获取失败。
- `-2` dev_handle 为无效句柄。
- `-3` tracking_state 为 nullptr。
- `-4` dev_id 为无效设备识别号。

[基础能力规范] 设备跟踪系统状态改变事件回调

[功能实现]

- GSXR_DEV_EVENT_CALLBACK_TYPE_TRACKING_STATE_CHANGED 设备跟踪系统状态改变时，XR 设备插件调用事件回调函数通知 XR Runtime，param1 为设备识别号(gsxr_dev_id_t)，param2 为新跟踪状态(gsxr_tracking_state_t)，param3 为前跟踪状态(gsxr_tracking_state_t)。

8.2. 跟踪模式

XR 设备的空间跟踪系统可能只具备旋转跟踪能力，也可能兼具位置跟踪能力。XR Runtime 可经由 get_tracking_mode 函数获取跟踪系统当前的跟踪模式，XR 设备插件必须确实输出当前跟踪模式供 XR Runtime 获取。

[枚举] gsxr_tracking_mode_t (跟踪算法定位模式)

名称	数值	描述
GSXR_TRACKING_MODE_NONE	0	无跟踪能力
GSXR_TRACKING_MODE_ROTATIONAL	0x1	具备旋转跟踪能力
GSXR_TRACKING_MODE_POSITIONAL	0x2	具备位置跟踪能力

[函数] gsxr_dev_tracking_ops_t (*get_tracking_mode) (获取跟踪模式)

```
int32_t (*get_tracking_mode)(
    gsxr_dev_handle_t                dev_handle,
    gsxr_dev_id_t                    dev_id,
    uint32_t*                        current_mode,
    uint32_t*                        supported_mode);
```

说明：

获取设备跟踪模式

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] current_mode 当前跟踪模式，为 gsxr_tracking_mode_t 的位掩码

[out] supported_mode 设备支持的所有跟踪模式，为 gsxr_tracking_mode_t 的位掩码

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

3DoF 跟踪模式为 GSXR_TRACKING_MODE_ROTATIONAL，6DoF 跟踪模式为 (GSXR_TRACKING_MODE_ROTATIONAL | GSXR_TRACKING_MODE_POSITIONAL)

[基础能力规范] gsxr_dev_tracking_ops_t (*get_tracking_mode)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `current_mode` 必须是指向 uint32_t 数值的有效指针。
- `supported_mode` 必须是指向 uint32_t 数值的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备当前的跟踪模式填入 current_mode 中输出。所有支持的跟踪模式填入 supported_mode 输出。

[返回值]

成功

- `0` 设备跟踪模式获取成功。

失败

- -1 设备跟踪模式获取失败。
- -2 dev_handle 为无效句柄。
- -3 current_mode 为 nullptr。或 supported_mode 为 nullptr。
- -4 dev_id 为无效设备识别号。

[基础能力规范] 设备跟踪模式变换事件回调

[功能实现]

- GSXR_DEV_EVENT_CALLBACK_TYPE_TRACKING_MODE_CHANGED 设备跟踪模式发生变换时，XR 设备插件调用事件回调函数通知 XR Runtime，param1 为设备识别号(gsxr_dev_id_t)，param2 为新跟踪模式(gsxr_tracking_mode_t)，param3 为前跟踪模式(gsxr_tracking_mode_t)。

8.3. 开启与停止跟踪系统

XR Runtime 可经由 start_tracking 及 stop_tracking 函数，控制 XR 设备跟踪系统的使用时机以及将要使用的跟踪模式。

[函数] gsxr_dev_tracking_ops_t (*start_tracking) (开启跟踪系统)

```
int32_t (*start_tracking)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    uint32_t                        tracking_mode);
```

说明：

开启设备跟踪系统

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn

获取

[in] tracking_mode 设置开启的跟踪模式，为 gsxr_tracking_mode_t 的位掩码

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为

GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*start_tracking)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `tracking_mode` 必须是有效的跟踪模式数值位掩码。

[函数实现]

- XR 设备插件根据输入的 tracking_mode 跟踪模式起始识别号 dev_id 设备的跟踪系统。

[返回值]

成功

- `0` 设备跟踪系统起始成功。

失败

- `-1` 设备跟踪系统起始失败。
- `-2` dev_handle 为无效句柄。
- `-3` tracking_mode 为无效或设备不支持的跟踪模式。
- `-4` dev_id 为无效设备识别号。

[函数] gsxr_dev_tracking_ops_t (*stop_tracking) (停止跟踪系统)

```
int32_t (*stop_tracking)(
```

```
    gsxr_dev_handle_t                   dev_handle,
    gsxr_dev_id_t                       dev_id);
```

说明：

停止设备跟踪系统

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*stop_tracking)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。

[函数实现]

- XR 设备插件停止识别号 dev_id 设备的跟踪系统。

[返回值]

成功

- `0` 设备跟踪系统停止成功。

失败

- `-1` 设备跟踪系统停止失败。
- `-2` dev_handle 为无效句柄。
- `-4` dev_id 为无效设备识别号。

8.4. 获取跟踪姿态数据

在成功起始 XR 设备的空间跟踪系统后， XR Runtime 可经由 get_pose_data 函数获取当前跟踪系统的跟踪姿态数据，XR 设备插件必须确实输出当前跟踪姿态供 XR Runtime 获取。

[结构体] gsxr_pose_data_t (设备跟踪姿态数据)

```
typedef struct gsxr_pose_data {
    int64_t             timestamp;
    bool                pose_valid;
    gsxr_posef_t        pose;
    bool                velocity_valid;
    gsxr_velocityf_t    velocity;
} gsxr_pose_data_t;
```

说明：

位姿及速度数据结构体

成员：

timestamp 姿态时间戳

pose_valid true 表示位姿有效, false 表示位姿无效

pose 位姿结构体

velocity_valid true 表示速度有效, false 表示速度无效

velocity 速度结构体

[函数] gsxr_dev_tracking_ops_t (*get_pose_data) (获取跟踪姿态数据)

```
int32_t (*get_pose_data)(
     gsxr_dev_handle_t                dev_handle,
     gsxr_dev_id_t                    dev_id,
     gsxr_pose_data_t*                pose_data);
```

说明：

获取设备姿态数据

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] pose_data 指向 gsxr_pose_data_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*get_pose_data)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `pose_data` 必须是指向 gsxr_pose_data_t 结构的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的跟踪姿态数据填入 pose_data 中输出。

[返回值]

成功

- `0` 设备跟踪姿态数据获取成功。

失败

- `-1` 设备跟踪姿态数据获取失败。
- `-2` dev_handle 为无效句柄。
- `-3` pose_data 为 nullptr。
- `-4` dev_id 为无效设备识别号。

8.5. 获取传感器数据

除跟踪姿态数据外， XR Runtime 也可经由 get_sensor_data 函数获取传感器数据，传感器数据通常用于特定的校正算法使用，一般应用无需传感器数据。

[结构体] gsxr_sensor_data_t (设备传感器数据)

```
typedef struct gsxr_sensor_data {
    int64_t             gyro_ts;
    float               gyro_x;
    float               gyro_y;
    float               gyro_z;
    int64_t             acc_ts;
    float               acc_x;
    float               acc_y;
    float               acc_z;
    int64_t             mag_ts;
    float               mag_x;
    float               mag_y;
    float               mag_z;
} gsxr_sensor_data_t;
```

说明：

传感器数据结构体

成员：

gyro_ts 陀螺仪数据时间戳

gyro_x x 轴旋转弧度，单位 rad/s

gyro_y y 轴旋转弧度，单位 rad/s

gyro_z z 轴旋转弧度，单位 rad/s

acc_ts 加速度计数据时间戳

acc_x x 轴加速度，单位 m/s^2

acc_y y 轴加速度，单位 m/s^2

acc _z z 轴加速度，单位 m/s^2

mag_ts 磁力计数据时间戳

mag_x x 轴磁力强度，单位 uT

mag_y y 轴磁力强度，单位 uT

mag_z z 轴磁力强度，单位 uT

[函数] gsxr_dev_tracking_ops_t (*get_sensor_data) (获取传感器数据)

```
int32_t (*get_sensor_data)(
    gsxr_dev_handle_t               dev_handle,
    gsxr_dev_id_t                   dev_id,
    gsxr_sensor_data_t*             sensor_data);
```

说明：

获取设备传感器数据

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] sensor_data 指向 gsxr_sensor_data_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*get_sensor_data)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。
- sensor_data 必须是指向 gsxr_sensor_data_t 结构的有效指针。

[函数实现]

- XR 设备插件获取识别号 dev_id 设备的传感器数据填入 sensor_data 中输出。

[返回值]

成功

- 0 设备传感器数据获取成功。

失败

- -1 设备传感器数据获取失败。
- -2 dev_handle 为无效句柄。
- -3 sensor_data 为 nullptr。
- -4 dev_id 为无效设备识别号。

[函数指针定义] gsxr_dev_sensor_data_callback_fn(传感器数据回调函数)

typedef void(*gsxr_dev_sensor_data_callback_fn)(gsxr_sensor_data_t sensor_data,void* cookie);

说明：

传感器数据回调函数，当收到新的传感器数据时回调，获取最新的传感器数据与时间戳参数：

[in]sensor_data 传感器数据以及对应时间戳

[in]cookie 函数回调时带回的预备数值

返回值：无

备注：无

[函数] gsxr_dev_tracking_ops_t (*set_sensor_data_callback) (设置传感器数据的回调函数)

```
int32_t (*set_sensor_data_callback)(
    gsxr_dev_handle_t                dev_handle,
    gsxr_dev_id_t                    dev_id,
```

```
    gsxr_dev_sensor_data_callback_fn*  callback,
    void*                              cookie);
```

说明：

设置设备传感器数据的回调函数

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] callback 传感器数据回调函数指针

[cookie]函数回调时带回的预备数值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*set_sensor_data_callback)

[参数检验]

- `dev_handle` 必须是有效的设备句柄。
- `dev_id` 必须是有效的设备识别号。
- `callback` 必须是 gsxr_dev_sensor_data_callback_fn 函数类型的有效指针。

[函数实现]

- XR 设备插件记录 XR Runtime 设置的 gsxr_dev_sensor_data_callback_fn 函数指针，并在接收到新的传感器数据后回调。

[返回值]

成功

- `0` 设备传感器数据获取成功。

失败

- -1 设备传感器数据获取失败。
- -2 hmd_handle 为无效句柄。
- -3 callback 为 nullptr。
- -4 hmd_id 为无效设备识别号。

8.6. 重置跟踪系统原点

在某些特殊的使用条件下， XR Runtime 必须以当前的位姿，重置跟踪系统原点，使该设备之后的跟踪姿态以此新置原点重新计算，XR Runtime 可经由函数 relocalize_origin 重置设备跟踪系统原点。

[函数] gsxr_dev_tracking_ops_t (*relocalize_origin) (重置跟踪原点)

```
int32_t (*relocalize_origin)(
    gsxr_dev_handle_t                dev_handle,
    gsxr_dev_id_t                    dev_id);
```

说明：

重置设备跟踪系统原点

参数：

[in] dev_handle 设备句柄，由 open 函数获取

[in] dev_id 设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为

GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_tracking_ops_t (*relocalize_origin)

[参数检验]

- dev_handle 必须是有效的设备句柄。
- dev_id 必须是有效的设备识别号。

[函数实现]

- XR 设备插件重置识别号 dev_id 设备的跟踪系统原点。

[返回值]

成功

- 0 设备跟踪系统圆点重置成功。

失败

- -1 设备跟踪系统圆点重置失败。
- -2 dev_handle 为无效句柄。
- -4 dev_id 为无效设备识别号。

9. XR 头戴式显示器函数说明

GSXR 通用设备接口将 XR 头显设备函数定义于 gsxr_dev_hmd_ops_t 结构之中，除包含前面章节所述的通用函数(详见第 5 章)、信息函数(详见第 6 章)、输入函数(详见第 7 章)、及跟踪函数(详见第 8 章)外，同样也定义了头显设备的专用函数。

```
typedef struct gsxr_dev_hmd_ops {
    gsxr_dev_common_ops_t common;
    gsxr_dev_info_ops_t info;
    gsxr_dev_input_ops_t input;
    gsxr_dev_tracking_ops_t tracking;
```

```
int32_t (*get_hmd_configurations)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        gsxr_hmd_configurations_t* configs);
int32_t (*get_hmd_ipd)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        float* ipd);
int32_t (*set_vsync_callback)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        gsxr_hmd_vsync_callback_fn callback,
        void* cookie);
int32_t (*set_proximity_callback)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        gsxr_hmd_proximity_callback_fn callback,
        void* cookie);
int32_t (*set_ipd_callback)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        gsxr_hmd_ipd_callback_fn callback,
        void* cookie);
int32_t (*get_stream_volume)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        uint32_t* current_volume,
        uint32_t* max_volume);
int32_t (*set_stream_volume)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        uint32_t volume);
int32_t (*get_screen_brightness)(
        gsxr_dev_handle_t hmd_handle,
        gsxr_dev_id_t hmd_id,
        uint32_t* current_brightness,
```

```
                uint32_t* max_brightness);
    int32_t (*set_screen_brightness)(
                gsxr_dev_handle_t hmd_handle,
                gsxr_dev_id_t hmd_id,
                uint32_t brightness);
} gsxr_dev_hmd_ops_t;
```

9.1. 获取头显设备实例

XR Runtime 在操作 XR 头显设备插件所实现的 gsxr_dev_hmd_ops_t 成员函数之前，必须先调用 gsxr_dev_get_hmd_instance 函数获取头显设备实例结构体 gsxr_dev_hmd_t，实例结构中除可获取头显插件所实现的 gsxr_dev_hmd_ops_t 所有成员函数指针外，同样可以获取 GSXR 的 API 版本号，XR Runtime 必须以此实例为进入点开始 XR 头显设备的操作。

[结构体] gsxr_dev_hmd_t (头显设备实例)

```
typedef struct gsxr_dev_hmd {
    gsxr_version            api_version;
    gsxr_dev_hmd_ops_t*     hmd_ops;
} gsxr_dev_hmd_t;
```

说明：

头显设备实例结构体

成员：

api_version 头显实例使用的 API 版本号 GSXR_DEVICE_CURRENT_API_VERSION

hmd_ops 头显实例实现的 GSXR 头显设备函数指针

[函数] gsxr_dev_get_hmd_instance (获取头显设备实例)

```
gsxr_dev_hmd_t* gsxr_dev_get_hmd_instance();
```

说明：

获取头显设备实例

参数：

无

返回值：

头显设备实例结构体 gsxr_dev_hmd_t

备注：

无

XR Runtime 在获取头显设备实例后，便可调用通用函数(*open)开启头显设备获取头显设备句柄(gsxr_dev_handle_t)，获取头显设备句柄后设置事件回调函数(*set_event_callback)，等待头显设备连接后 XR 头显设备插件回调连接事件 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，其回调参数 param1 即为头显设备识别号(gsxr_dev_id_t)。

9.2. 获取头显设备配置

XR 头显设备配置定义于结构体 gsxr_hmd_configurations_t 中，XR Runtime 可经由函数 get_hmd_configurations 获取，XR 头显设备插件必须根据实际头显硬件的类型及配置，正确填入所有头显配置数据供 XR Runtime 使用。

[结构体] gsxr_hmd_configurations_t (头显配置)

```
typedef struct gsxr_hmd_configurations {
    gsxr_hmd_type_t                 hmd_type;
    gsxr_view_type_t                view_type;
    gsxr_display_config_t           display_config;
```

```
    gsxr_lens_view_config_t         lens_view_config;
    gsxr_sensor_to_head_config_t    sensor_to_head_config;
} gsxr_hmd_configurations_t;
```

说明：

头显配置结构体

成员：

hmd_type 头显类型

view_type 视图类型

display_config 显示屏配置

lens_view_config 镜片视图配置

sensor_to_head_config 跟踪传感器偏移量配置（相对于视图中心）

[枚举] gsxr_hmd_type_t（头显类型）

名称	数值	描述
GSXR_HMD_TYPE_VR	1	VR 头显
GSXR_HMD_TYPE_AR	2	AR 头显

[枚举] gsxr_view_type_t（视图类型）

名称	数值	描述
GSXR_VIEW_TYPE_MONO	1	单视图
GSXR_VIEW_TYPE_STEREO	2	双视图

[枚举] gsxr_scanout_order_t（屏幕扫描输出顺序）

名称	数值	描述
GSXR_SCANOUT_ORDER_LEFT_TO_RIGHT	1	左至右
GSXR_SCANOUT_ORDER_RIGHT_TO_LEFT	2	右至左
GSXR_SCANOUT_ORDER_TOP_TO_BOTTOM	3	上至下

GSXR_SCANOUT_ORDER_BOTTOM_TO_TOP	4	下至上

[结构体] gsxr_display_config_t (显示屏配置)

```
typedef struct gsxr_display_config {
    uint32_t                        width;
    uint32_t                        height;
    uint32_t                        refresh_rate;
    float                           vsync_fixed_offset;
    gsxr_scanout_order_t            scanout_order;
} gsxr_display_config_t;
```

说明：

显示屏配置结构体

成员：

width 屏幕宽度(pixel)

height 屏幕高度(pixel)

refresh_rate 刷新频率(Hz)

vsync_fixed_offset V-Sync 的固定偏移时间值(ms)

scanout_order 扫描输出顺序

[结构体] gsxr_frustum_t (视锥体)

```
typedef struct gsxr_frustum {
    float          left;
    float          right;
    float          bottom;
    float          top;
} gsxr_frustum_t;
```

说明：

视锥体结构体

成员：

left 视锥体左边界

right 视锥体右边界

bottom 视锥体下边界

top 视锥体上边界

[结构体] gsxr_chromatic_poly_t (畸变校正参数)

```
typedef struct gsxr_chromatic_poly {
    float       red[8];
    float       green[8];
    float       blue[8];
} gsxr_chromatic_poly_t;
```

说明：

畸变校正参数结构体

成员：

red 红畸变校正参数数组

green 绿畸变校正参数数组

blue 蓝畸变校正参数数组

[结构体] gsxr_lens_view_config_t (镜片视图配置)

```
typedef struct gsxr_lens_view_config {
    gsxr_frustum_t              left_frustum;
    gsxr_frustum_t              right_frustum;
    float                       left_eye_center[2];
    float                       right_eye_center[2];
    gsxr_chromatic_poly_t       left_chromatic_poly;
    gsxr_chromatic_poly_t       right_chromatic_poly;
    uint32_t                    warp_mesh_columns;
    uint32_t                    warp_mesh_rows;
} gsxr_lens_view_config_t;
```

说明：

镜片视图配置结构体

成员：

left_frustum 左眼视锥体

right_frustum 右眼视锥体

left_eye_center 左眼畸变校正中心(UVs)

right_eye_center 右眼畸变校正中心(UVs)

left_chromatic_poly 左眼畸变校正参数

right_chromatic_poly 右眼畸变校正参数

warp_mesh_columns 畸变网格 column 数目

warp_mesh_rows 畸变网格 row 数目

[结构体] gsxr_sensor_to_head_config_t (头显跟踪传感器偏移量配置)

```
typedef struct gsxr_sensor_to_head_config {
    gsxr_vector3f_t             position_offest;
    gsxr_quaternionf_t          orientation_offset;
} gsxr_sensor_to_head_config_t;
```

说明：

头显跟踪传感器偏移量配置结构体

成员：

position_offest 位置偏移量

orientation_offset 方向偏移量

[函数] gsxr_dev_hmd_ops_t (*get_hmd_configurations) (获取头显配置)

```
int32_t (*get_hmd_configurations)(
     gsxr_dev_handle_t                  hmd_handle,
     gsxr_dev_id_t                      hmd_id,
     gsxr_hmd_configurations_t*         configs);
```

说明：

获取头显设备配置

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数

gsxr_dev_event_callback_fn 获取

[out] configs 指向 gsxr_hmd_configurations_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为

GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*get_hmd_configurations)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。
- `hmd_id` 必须是有效的设备识别号。
- `configs` 必须是指向 gsxr_hmd_configurations_t 结构的有效指针。

[函数实现]

- XR 头显设备插件获取识别号 hmd_id 设备的头显设备配置信息，填入 configs 中输出。

[返回值]

成功

- `0` 头显设备配置获取成功。

失败

- `-1` 头显设备配置获取失败。
- `-2` hmd_handle 为无效句柄。
- `-3` configs 为 nullptr。
- `-4` hmd_id 为无效设备识别号。

[函数] gsxr_dev_hmd_ops_t (*get_hmd_ipd) (获取 IPD 距离)

```
int32_t (*get_hmd_ipd)(
```

```
    gsxr_dev_handle_t                   hmd_handle,
    gsxr_dev_id_t                       hmd_id,
    float*                              ipd);
```

说明：

获取头显双眼 IPD 距离

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] ipd 双眼 IPD 距离，单位为米

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*get_hmd_ipd)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。
- `hmd_id` 必须是有效的设备识别号。
- `ipd` 必须是指向 float 数值的有效指针。

[函数实现]

- XR 头显设备插件获取识别号 hmd_id 设备当前的双眼 IPD 距离(单位为米)，填入 ipd 中输出。

[返回值]

成功

- `0` 头显设备 IPD 获取成功。

失败

- -1 头显设备 IPD 获取失败。
- -2 hmd_handle 为无效句柄。
- -3 ipd 为 nullptr。
- -4 hmd_id 为无效设备识别号。

9.3. 设置头显设备回调函数

XR Runtime 可经由设置回调函数指针给 XR 设备插件，XR 设备插件在对应的设备信号或状态发生时必须回调对应的函数指针通知 XR Runtime。XR Runtime 开发回调函数必须考虑回调线程的即时性，不可于回调函数中执行过于耗时的 Runtime 工作。

1. 屏幕垂直同步回调函数 - 当显示屏幕发生 V-Sync 信号时回调

[类型定义] gsxr_hmd_vsync_callback_fn (头显屏幕垂直同步回调函数)

```
typedef void (*gsxr_hmd_vsync_callback_fn) (
      int64_t timestamp,
      void* cookie);
```

说明：

头显屏幕垂直同步回调函数，当屏幕发生 V-Sync 信号时回调，回调频率与屏幕帧率同步

参数：

[in] timestamp 垂直同步信号时间戳

[in] cookie 函数回调时带回的预备数值

返回值：

无

备注：

无

[函数] gsxr_dev_hmd_ops_t (*set_vsync_callback) (设置 V-Sync 回调函数)

```
int32_t (*set_vsync_callback)(
    gsxr_dev_handle_t                hmd_handle,
    gsxr_dev_id_t                    hmd_id,
    gsxr_hmd_vsync_callback_fn       callback,
    void*                            cookie);
```

说明：

设置头显屏幕垂直同步信号回调函数

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] callback 头显屏幕垂直同步信号回调函数

[in] cookie 函数回调时带回的预备数值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的回调函数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*set_vsync_callback)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。
- `hmd_id` 必须是有效的设备识别号。
- `callback` 必须是 gsxr_hmd_vsync_callback_fn 函数类型的有效指针。

[函数实现]

- XR 头显设备插件记录 XR Runtime 设置的 gsxr_hmd_vsync_callback_fn 函数指针，并在头显 V-Sync 信号发生时回调函数。

[返回值]

成功

- 0 回调函数设置成功。

失败

- -1 回调函数设置失败。
- -2 hmd_handle 为无效句柄。
- -3 callback 为 nullptr。
- -4 hmd_id 为无效设备识别号。

2. 头部传感器回调函数 - 当头显的距离传感器侦测头部距离发生变动时回调

[类型定义] gsxr_hmd_proximity_callback_fn（头部距离传感器回调函数）

```
typedef void (*gsxr_hmd_proximity_callback_fn) (
    float distance,
    void* cookie);
```

说明：

头部距离传感器回调函数，当头部与传感器距离发生变动时回调

参数：

[in] distance 头部与距离传感器距离，单位为公分

[in] cookie 函数回调时带回的预备数值

返回值：

无

备注：

无

[函数] gsxr_dev_hmd_ops_t (*set_proximity_callback)（设置头部距离传感器回调函数）

```
int32_t (*set_proximity_callback)(
```

```
    gsxr_dev_handle_t                   hmd_handle,
    gsxr_dev_id_t                       hmd_id,
    gsxr_hmd_proximity_callback_fn      callback,
    void*                               cookie);
```

说明：

设置头显距离传感器信号回调函数

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] callback 头显距离传感器信号回调函数

[in] cookie 函数回调时带回的预备数值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的回调函数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*set_proximity_callback)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。
- `hmd_id` 必须是有效的设备识别号。
- `callback` 必须是 gsxr_hmd_proximity_callback_fn 函数类型的有效指针。

[函数实现]

- XR 头显设备插件记录 XR Runtime 设置的 gsxr_hmd_proximity_callback_fn 函数指针，并在传感器侦测到头部与头显距离发生变动时回调函数。

[返回值]

成功

- 0 回调函数设置成功。

失败

- -1 回调函数设置失败。
- -2 hmd_handle 为无效句柄。
- -3 callback 为 nullptr。
- -4 hmd_id 为无效设备识别号。

3. 头显 IPD 回调函数 - 当头显镜片的距离发生变动时回调

[类型定义] gsxr_hmd_ipd_callback_fn (头显 IPD 回调函数)

```
typedef void (*gsxr_hmd_ipd_callback_fn) (
      float ipd,
      void* cookie);
```

说明：

头显 IPD 回调函数, 当头显 IPD 变动时回调

参数：

[in] ipd 头显双眼 IPD 距离, 单位为米

[in] cookie 函数回调时带回的预备数值

返回值：

无

备注：

无

[函数] gsxr_dev_hmd_ops_t (*set_ipd_callback) (设置 IPD 回调函数)

```
int32_t (*set_ipd_callback)(
      gsxr_dev_handle_t                hmd_handle,
      gsxr_dev_id_t                    hmd_id,
      gsxr_hmd_ipd_callback_fn         callback,
      void*                            cookie);
```

说明：

设置头显 IPD 变动回调函数

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数

gsxr_dev_event_callback_fn 获取

[in] callback 头显 IPD 变动回调函数

[in] cookie 函数回调时带回的预备数值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的回调函数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为

GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*set_ipd_callback)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。
- `hmd_id` 必须是有效的设备识别号。
- `callback` 必须是 gsxr_hmd_ipd_callback_fn 函数类型的有效指针。

[函数实现]

- XR 头显设备插件记录 XR Runtime 设置的 gsxr_hmd_ipd_callback_fn 函数指针，并在头显镜片 IPD 距离发生变动时回调函数。

[返回值]

成功

- `0` 回调函数设置成功。

失败

- `-1` 回调函数设置失败。
- `-2` hmd_handle 为无效句柄。
- `-3` callback 为 nullptr。

- -4 hmd_id 为无效设备识别号。

9.4. 获取与设置头显音量状态

头显音量状态可经由 get_stream_volume 及 set_stream_volume 获取及设置。

[函数] gsxr_dev_hmd_ops_t (*get_stream_volume) (获取头显音量状态)

```
int32_t (*get_stream_volume)(
    gsxr_dev_handle_t               hmd_handle,
    gsxr_dev_id_t                   hmd_id,
    uint32_t*                       current_volume,
    uint32_t*                       max_volume);
```

说明：

获取头显设备音量状态

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数

gsxr_dev_event_callback_fn 获取

[out] current_volume 当前音量值，0 <= current_volume <= max_volume

[out] max_volume 最大音量值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为

GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*get_stream_volume)

[参数检验]

- hmd_handle 必须是有效的设备句柄。
- hmd_id 必须是有效的设备识别号。
- current_volume 必须是指向 uint32_t 数值的有效指针。
- max_volume 必须是指向 uint32_t 数值的有效指针。

[函数实现]

- XR 头显设备插件获取头显设备当前的音量值填入 current_volume，并将最大音量值填入 max_volume 中输出。

[返回值]

成功

- 0 头显音量获取成功。

失败

- -1 头显音量获取失败。
- -2 hmd_handle 为无效句柄。
- -3 current_volume 为 nullptr。或 max_volume 为 nullptr。
- -4 hmd_id 为无效设备识别号。

[函数] gsxr_dev_hmd_ops_t (*set_stream_volume) (设置头显音量状态)

```
int32_t (*set_stream_volume)(
    gsxr_dev_handle_t               hmd_handle,
    gsxr_dev_id_t                   hmd_id,
    uint32_t                        volume);
```

说明：

设置头显设备音量状态

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] volume 设置音量值, 音量值需介于 [0 - 最大音量值] 区间

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*set_stream_volume)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。
- `hmd_id` 必须是有效的设备识别号。
- `volume` 必须是有效的音量数值。

[函数实现]

- XR 头显设备插件调整头显设备的音量数值为 volume。

[返回值]

成功

- `0` 头显音量设置成功。

失败

- `-1` 头显音量设置失败。
- `-2` hmd_handle 为无效句柄。
- `-3` volume 不介于 [0 - 最大音量值] 区间。
- `-4` hmd_id 为无效设备识别号。

9.5. 获取与设置头显屏幕亮度

头显屏幕亮度可经由 get_screen_brightness 及 set_screen_brightness 获

取及设置。

[函数] gsxr_dev_hmd_ops_t (*get_screen_brightness) (获取头显屏幕亮度)

```
int32_t (*get_screen_brightness)(
    gsxr_dev_handle_t                hmd_handle,
    gsxr_dev_id_t                    hmd_id,
    uint32_t*                        current_brightness,
    uint32_t*                        max_brightness);
```

说明：

获取头显设备屏幕亮度

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] current_brightness 当前屏幕亮度值，0 <= current_brightness <= max_brightness

[out] max_brightness 最大屏幕亮度

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*get_screen_brightness)

[参数检验]

- `hmd_handle` 必须是有效的设备句柄。

- hmd_id 必须是有效的设备识别号。
- current_brightness 必须是指向 uint32_t 数值的有效指针。
- max_brightness 必须是指向 uint32_t 数值的有效指针。

[函数实现]

- XR 头显设备插件获取头显设备当前的屏幕亮度填入 current_brightness，并将最大屏幕亮度填入 max_brightness 中输出。

[返回值]

成功

- 0 头显屏幕亮度获取成功。

失败

- -1 头显屏幕亮度获取失败。
- -2 hmd_handle 为无效句柄。
- -3 current_brightness 为 nullptr。或 max_brightness 为 nullptr。
- -4 hmd_id 为无效设备识别号。

[函数] gsxr_dev_hmd_ops_t (*set_screen_brightness) (设置头显屏幕亮度)

```
int32_t (*set_screen_brightness)(
    gsxr_dev_handle_t               hmd_handle,
    gsxr_dev_id_t                   hmd_id,
    uint32_t                        brightness);
```

说明：

设置头显设备屏幕亮度

参数：

[in] hmd_handle 头显设备句柄，由 open 函数获取

[in] hmd_id 头显设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] brightness 设置屏幕亮度值，屏幕亮度值需介于 [0 - 最大屏幕亮度值] 区间

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_hmd_ops_t (*set_screen_brightness)

[参数检验]

- hmd_handle 必须是有效的设备句柄。
- hmd_id 必须是有效的设备识别号。
- brightness 必须是有效的屏幕亮度数值。

[函数实现]

- XR 头显设备插件调整头显设备的屏幕亮度数值为 brightness。

[返回值]

成功

- 0 头显屏幕亮度设置成功。

失败

- -1 头显屏幕亮度设置失败。
- -2 hmd_handle 为无效句柄。
- -3 brightness 不介于 [0 - 最大屏幕亮度值] 区间。
- -4 hmd_id 为无效设备识别号。

10. XR 手柄控制器函数说明

GSXR 通用设备接口将 XR 手柄设备函数定义于 gsxr_dev_controller_ops_t 结构之中，除包含前面章节所述的通用函数(详见第 5 章)、信息函数(详见第 6 章)、输入函数(详见第 7 章)、及跟踪函数(详见第 8 章)外，同样也定义了手柄设备的专用函数。

```
typedef struct gsxr_dev_controller_ops {
    gsxr_dev_common_ops_t common;
    gsxr_dev_info_ops_t info;
    gsxr_dev_input_ops_t input;
    gsxr_dev_tracking_ops_t tracking;

    int32_t (*start_scan_controllers)(
                gsxr_dev_handle_t controller_handle);
    int32_t (*get_available_controllers)(
                gsxr_dev_handle_t controller_handle,
                gsxr_controller_list_t** controller_list);
    int32_t (*stop_scan_controllers)(
                gsxr_dev_handle_t controller_handle);
    int32_t (*connect)(
                gsxr_dev_handle_t controller_handle,
                uint64_t CON_ID);
    int32_t (*disconnect)(
                gsxr_dev_handle_t controller_handle,
                uint64_t CON_ID);
    int32_t (*get_paired_controllers)(
                gsxr_dev_handle_t controller_handle,
                gsxr_controller_list_t** controller_list);
    int32_t (*get_controller_configurations)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id,
                gsxr_controller_configurations_t* configs);
    int32_t (*get_touchpad_mode)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id,
                bool* enabled,
                uint32_t* current_mode,
                uint32_t* supported_mode);
    int32_t (*enable_touchpad)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id,
```

```
                uint32_t touchpad_mode);
    int32_t (*set_touchpad_mode)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id,
                uint32_t touchpad_mode);
    int32_t (*disable_touchpad)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id);
    int32_t (*trigger_vibration)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id,
                const gsxr_haptic_vibration_t* haptics);
    int32_t (*upgrade_firmware)(
                gsxr_dev_handle_t controller_handle,
                gsxr_dev_id_t controller_id,
                gsxr_controller_firmware_callback_fn callback,
                void* cookie);
} gsxr_dev_controller_ops_t;
```

10.1. 获取手柄设备实例

XR Runtime 在操作 XR 手柄设备插件所实现的 gsxr_dev_controller_ops_t 成员函数之前，必须先调用 gsxr_dev_get_controller_instance 函数获取手柄设备实例结构体 gsxr_dev_controller_t，实例结构中除可获取手柄插件所实现的 gsxr_dev_controller_ops_t 所有成员函数指针外，同样可获取 GSXR 的 API 版本号，XR Runtime 必须以此实例为进入点开始 XR 手柄设备的操作。

[结构体] gsxr_dev_controller_t (手柄设备实例)

```
typedef struct gsxr_dev_controller {
    gsxr_version                    api_version;
    gsxr_dev_controller_ops_t*      controller_ops;
```

```
} gsxr_dev_controller_t;
```

说明：

手柄设备实例结构体

成员：

api_version 手柄实例使用的 API 版本号 GSXR_DEVICE_CURRENT_API_VERSION

controller_ops 手柄实例实现的 GSXR 手柄设备函数指针

[函数] gsxr_dev_get_controller_instance (获取手柄设备实例)

```
gsxr_dev_controller_t* gsxr_dev_get_controller_instance();
```

说明：

获取手柄设备实例

参数：

无

返回值：

手柄设备实例结构体 gsxr_dev_controller_t

备注：

无

XR Runtime 在获取手柄设备实例后，便可调用通用函数(*open)开启手柄设备获取手柄设备句柄(gsxr_dev_handle_t)，获取手柄设备句柄后设置事件回调函数(*set_event_callback)，等待手柄设备连接后 XR 手柄设备插件回调连接事件 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，其回调参数 param1 即为手柄设备识别号(gsxr_dev_id_t)。

10.2. 手柄连线函数

XR Runtime 在开启手柄设备获取手柄设备句柄(gsxr_dev_handle_t)后，可

依据产品的手柄配对流程，在适当时机调用以下函数完成手柄的搜索、连线、及配对，XR 手柄设备插件必须依据函数需求实现相应的手柄连线行为。

[函数] gsxr_dev_controller_ops_t (*start_scan_controllers) (开始搜索手柄)

```
int32_t (*start_scan_controllers)(gsxr_dev_handle_t controller_handle);
```

说明：

开始搜索手柄

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

备注：

无

[基础能力规范] gsxr_dev_controller_ops_t (*start_scan_controllers)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。

[函数实现]

- XR 手柄设备插件依据产品手柄配对设计，开始搜索可用手柄。

[返回值]

成功

- `0` 成功开始手柄搜索。

失败

- `-1` 手柄无法开始搜索。
- `-2` controller_handle 为无效句柄。

[结构体] gsxr_controller_device_t (手柄控制器信息)

```
typedef struct gsxr_controller_device {
    uint64_t                        CON_ID;
    const char*                     name;
    bool                            paired;
    bool                            connected;
} gsxr_controller_device_t;
```

说明：

手柄控制器信息结构体

成员：

CON_ID 连线识别号

name 名称

paired 是否曾配对过

connected 是否已连线

[结构体] gsxr_controller_list_t (手柄控制器列表)

```
typedef struct gsxr_controller_list {
    uint32_t                        devices_len;
    gsxr_controller_device_t*       devices;
} gsxr_controller_list_t;
```

说明：

手柄控制器列表结构体

成员：

devices_len 数组个数

devices 指向手柄信息数组

[函数] gsxr_dev_controller_ops_t (*get_available_controllers) (获取搜索到的手柄列表)

```
int32_t (*get_available_controllers)(
    gsxr_dev_handle_t                controller_handle,
    gsxr_controller_list_t**         controller_list);
```

说明：

获取搜索到的手柄列表

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[out] controller_list 搜索到的手柄列表

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

此函数需介于 start_scan_controllers 及 stop_scan_controllers 之间调用

[基础能力规范] gsxr_dev_controller_ops_t (*get_available_controllers)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_list` 必须是指向 gsxr_controller_list_t 指针的有效指针。

[函数实现]

- XR 手柄设备插件依据搜索结果，将搜索到的手柄信息填入 controller_list 中输出。
- controller_list 内存只保留最后一次调用的手柄列表数据，调用 stop_scan_controllers 停止搜索手柄后，释放 controller_list 内存。

[返回值]

成功

- `0` 手柄列表获取成功。

失败

- `-1` 手柄列表获取失败。
- `-2` controller_handle 为无效句柄。

- -3 controller_list 为 nullptr。

[函数] gsxr_dev_controller_ops_t (*stop_scan_controllers) (停止搜索手柄)

```
int32_t (*stop_scan_controllers)(
    gsxr_dev_handle_t                controller_handle);
```

说明：

停止搜索手柄

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

备注：

无

[基础能力规范] gsxr_dev_controller_ops_t (*stop_scan_controllers)

[参数检验]

- controller_handle 必须是有效的设备句柄。

[函数实现]

- XR 手柄设备插件依据产品手柄配对设计，停止搜索可用手柄。

[返回值]

成功

- 0 成功停止手柄搜索。

失败

- -1 无法停止手柄搜索。
- -2 controller_handle 为无效句柄。

[函数] gsxr_dev_controller_ops_t (*connect) (连接手柄)

```
int32_t (*connect)(
    gsxr_dev_handle_t          controller_handle,
    uint64_t                   CON_ID);
```

说明：

连接手柄

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] CON_ID 手柄连线识别号，由搜索到的手柄信息 gsxr_controller_device_t 中获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_controller_ops_t (*connect)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `CON_ID` 必须是有效的手柄连线识别号。

[函数实现]

- XR 手柄设备插件依据产品手柄配对设计，连线配对 CON_ID 手柄。
- 连线配对成功后，回调 XR Runtime 设置的事件回调函数(gsxr_dev_event_callback_fn)，回调事件为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，回调参数 param1 为手柄设备识别号(gsxr_dev_id_t)。

[返回值]

成功

- `0` 手柄连线配对成功。

失败

- -1 手柄连线配对失败。
- -2 controller_handle 为无效句柄。
- -3 CON_ID 为无效手柄连线识别号。

[函数] gsxr_dev_controller_ops_t (*disconnect) (断开手柄)

```
int32_t (*disconnect)(
    gsxr_dev_handle_t          controller_handle,
    uint64_t                   CON_ID);
```

说明：

断开手柄

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] CON_ID 手柄连线识别号，由搜索到的手柄信息 gsxr_controller_device_t 中获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_controller_ops_t (*disconnect)

[参数检验]

- controller_handle 必须是有效的设备句柄。
- CON_ID 必须是有效的手柄连线识别号。

[函数实现]

- XR 手柄设备插件依据产品手柄配对设计，断开已连线的 CON_ID 手柄。

[返回值]

成功

- `0` 手柄连线断开成功。

失败

- `-1` 手柄连线断开失败。
- `-2` controller_handle 为无效句柄。
- `-3` CON_ID 为无效手柄连线识别号。或该手柄未连线。

[函数] gsxr_dev_controller_ops_t (*get_paired_controllers) (获取曾配对过的手柄列表)

```
int32_t (*get_paired_controllers)(
    gsxr_dev_handle_t                controller_handle,
    gsxr_controller_list_t**         controller_list);
```

说明：

获取曾配对过的手柄列表

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[out] controller_list 曾配对过的手柄列表

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_controller_ops_t (*get_paired_controllers)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_list` 必须是指向 gsxr_controller_list_t 指针的有效指针。

[函数实现]

- XR 手柄设备插件依据手柄配对履历，将曾配对过的手柄信息填入 controller_list 中输

出。

- controller_list 内存只保留最后一次调用的手柄列表数据，调用 close 关闭手柄设备后，释放 controller_list 内存。

[返回值]

成功

- 0 手柄列表获取成功。

失败

- -1 手柄列表获取失败。
- -2 controller_handle 为无效句柄。
- -3 controller_list 为 nullptr。

10.3. 获取手柄设备配置

XR 手柄设备配置定义于结构体 gsxr_controller_configurations_t 中，XR Runtime 可经由函数 get_controller_configurations 获取，XR 手柄设备插件必须根据实际手柄硬件的类型及配置，正确填入所有手柄配置数据供 XR Runtime 使用。

[结构体] gsxr_controller_configurations_t (手柄配置)

```
typedef struct gsxr_controller_configurations {
    gsxr_controller_type_t          controller_type;
} gsxr_controller_configurations_t;
```

说明：

手柄控制器配置结构体

成员：

controller_type 手柄控制器类型

[枚举] gsxr_controller_type_t (手柄控制器类型)

名称	数值	描述
GSXR_CONTROLLER_TYPE_RIGHT	1	右手手柄控制器
GSXR_CONTROLLER_TYPE_LEFT	2	左手手柄控制器

[函数] gsxr_dev_controller_ops_t (*get_controller_configurations) (获取手柄配置)

```
int32_t (*get_controller_configurations)(
    gsxr_dev_handle_t                   controller_handle,
    gsxr_dev_id_t                       controller_id,
    gsxr_controller_configurations_t*   configs);
```

说明：

获取手柄设备配置

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] configs 指向 gsxr_controller_configurations_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*get_controller_configurations)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_id` 必须是有效的设备识别号。
- `configs` 必须是指向 gsxr_controller_configurations_t 结构的有效指针。

[函数实现]

- XR 手柄设备插件获取识别号 controller_id 设备的手柄设备配置信息，填入 configs 中输出。

[返回值]

成功

- `0` 手柄设备配置获取成功。

失败

- `-1` 手柄设备配置获取失败。
- `-2` controller_handle 为无效句柄。
- `-3` configs 为 nullptr。
- `-4` controller_id 为无效设备识别号。

10.4. 操控手柄触控板

若手柄触控板具备动态始能禁用及动态模式切换能力，XR 手柄插件可实现本节所述的对应函数供 XR Runtime 于必要时使用。

[枚举] gsxr_touchpad_mode_t (触控板操作模式)

名称	数值	描述
GSXR_TOUCHPAD_MODE_SWIPE	0x1	滑动模式
GSXR_TOUCHPAD_MODE_CLICK	0x2	按键模式
GSXR_TOUCHPAD_MODE_ANALOG	0x4	鼠标模式

[基础能力规范] 触控板滑动事件回调

[功能实现]

- 当触控板始能滑动模式时，若触控板判读到有效的滑动输入行为，XR 手柄插件调用事件回调函数通知 XR Runtime，param1 为设备识别号(gsxr_dev_id_t)，param2 为输入部件识别号(gsxr_input_id_t)。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_LEFT_TO_RIGHT_SWIPED 左至右滑动。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_RIGHT_TO_LEFT_SWIPED 右至左滑动。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_TOP_TO_BOTTOM_SWIPED 上至下滑动。
- GSXR_DEV_EVENT_CALLBACK_TYPE_INPUT_BOTTOM_TO_TOP_SWIPED 下至上滑动。

[函数] gsxr_dev_controller_ops_t (*get_touchpad_mode) (获取触控模式)

```
int32_t (*get_touchpad_mode)(
    gsxr_dev_handle_t                controller_handle,
    gsxr_dev_id_t                    controller_id,
    bool*                            enabled,
    uint32_t*                        current_mode,
    uint32_t*                        max_mode);
```

说明：

获取手柄设备触控模式

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[out] enabled touchpad 是否已使能

[out] current_mode 当前触控模式，为 gsxr_touchpad_mode_t 的位掩码

[out] supported_mode 设备支持的所有触控模式，为 gsxr_touchpad_mode_t 的位掩码

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED, param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*get_touchpad_mode)

[参数检验]

- controller_handle 必须是有效的设备句柄。
- controller_id 必须是有效的设备识别号。
- enabled 必须是指向 bool 数值的有效指针。
- current_mode 必须是指向 uint32_t 数值的有效指针。
- supported_mode 必须是指向 uint32_t 数值的有效指针。

[函数实现]

- XR 手柄插件获取手柄设备当前的起始状态填入 enabled，当前触控模式填入 current_mode，并将手柄支持的所有触控模式填入 supported_mode 中输出。

[返回值]

成功

- 0 触控模式获取成功。

失败

- -1 触控模式获取失败。
- -2 controller_handle 为无效句柄。
- -3 enabled 为 nullptr。current_mode 为 nullptr。或 supported_mode 为 nullptr。
- -4 controller_id 为无效设备识别号。

[函数] gsxr_dev_controller_ops_t (*enable_touchpad) (初始化触控板)

```
int32_t (*enable_touchpad)(
    gsxr_dev_handle_t         controller_handle,
    gsxr_dev_id_t             controller_id,
    uint32_t                  touchpad_mode);
```

说明：

使能手柄设备触控板

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数

gsxr_dev_event_callback_fn 获取

[in] touchpad_mode 设置使能的触控模式，为 gsxr_touchpad_mode_t 的位掩码

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为

GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*enable_touchpad)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_id` 必须是有效的设备识别号。
- `touchpad_mode` 必须是有效的触控模式位掩码。

[函数实现]

- XR 手柄插件根据 touchpad_mode 的触控模式始能触控板。

[返回值]

成功

- 0 触控板初始化成功。

失败

- -1 触控板初始化失败。
- -2 controller_handle 为无效句柄。
- -3 touchpad_mode 为无效触控模式。或手柄设备不支持的触控模式。
- -4 controller_id 为无效设备识别号。

[函数] gsxr_dev_controller_ops_t (*set_touchpad_mode) (设置触控模式)

```
int32_t (*set_touchpad_mode)(
    gsxr_dev_handle_t           controller_handle,
    gsxr_dev_id_t               controller_id,
    uint32_t                    touchpad_mode);
```

说明：

设置手柄设备触控模式

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] touchpad_mode 设置转换的触控模式，为 gsxr_touchpad_mode_t 的位掩码

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*set_touchpad_mode)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_id` 必须是有效的设备识别号。
- `touchpad_mode` 必须是有效的触控模式位掩码。

[函数实现]

- XR 手柄插件根据设置的 touchpad_mode 转换手柄触控模式。

[返回值]

成功

- 0 触控模式设置成功。

失败

- -1 触控模式设置失败。
- -2 controller_handle 为无效句柄。
- -3 touchpad_mode 为无效触控模式。或手柄设备不支持的触控模式。
- -4 controller_id 为无效设备识别号。

[函数] gsxr_dev_controller_ops_t (*disable_touchpad) (禁用触控板)

```
int32_t (*disable_touchpad)(
    gsxr_dev_handle_t                controller_handle,
    gsxr_dev_id_t                    controller_id);
```

说明：

禁用手柄设备触控板

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*disable_touchpad)

[参数检验]

- controller_handle 必须是有效的设备句柄。
- controller_id 必须是有效的设备识别号。

[函数实现]

- XR 手柄插件禁用触控板，XR Runtime 将无法使用触控板及其滑动事件。

[返回值]

成功

- 0 触控板禁用成功。

失败

- -1 触控板禁用失败。
- -2 controller_handle 为无效句柄。
- -4 controller_id 为无效设备识别号。

10.5. 触发手柄振动

手柄振动反馈可经由 trigger_vibration 函数触发。

[结构体] gsxr_haptic_vibration_t (振动反馈)

```
typedef struct gsxr_haptic_vibration {
    int64_t                         duration;
    uint32_t                        frequency;
    float                           intensity;
} gsxr_haptic_vibration_t;
```

说明：

振动反馈结构体

成员：

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_dev_controller_ops_t (*trigger_vibration) (触发振动反馈)

```
int32_t (*trigger_vibration)(
```

```
    gsxr_dev_handle_t                controller_handle,
    gsxr_dev_id_t                    controller_id,
    const gsxr_haptic_vibration_t*   haptics);
```

说明：

触发手柄振动反馈

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] haptics 指向 gsxr_haptic_vibration_t 结构体

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*trigger_vibration)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_id` 必须是有效的设备识别号。
- `haptics` 必须是指向 gsxr_haptic_vibration_t 结构的有效指针。

[函数实现]

- XR 手柄插件根据 haptics 的设置触发手柄振动反馈。

[返回值]

成功

- `0` 振动反馈触发成功。

失败

- -1 振动反馈触发失败。
- -2 controller_handle 为无效句柄。
- -3 haptics 为 nullptr 或无效的数值。
- -4 controller_id 为无效设备识别号。

10.6. 升级手柄固件

手柄固件可经由 upgrade_firmware 函数起动升级，XR Runtime 必须设置回调函数 gsxr_controller_firmware_callback_fn 获取固件更新状态及进度。

[类型定义] gsxr_controller_firmware_callback_fn (固件回调函数)

```
typedef void (*gsxr_controller_firmware_callback_fn) (
      gsxr_dev_id_t controller_id,
      int32_t status,
      int32_t progress,
      int32_t error,
      void* cookie);
```

说明：

手柄固件回调函数，当手柄升级固件版本时回调

参数：

[in] controller_id 手柄设备识别号

[in] status 更新状态，0 为无可被更新的固件，1 为更新中，2 为更新成功，3 为更新失败

[in] progress 更新进度，0 - 100

[in] error 错误代码

[in] cookie 函数回调时带回的预备数值

返回值：

无

备注：

无

[函数] gsxr_dev_controller_ops_t (*upgrade_firmware) (升级固件)

```
int32_t (*upgrade_firmware)(
    gsxr_dev_handle_t                       controller_handle,
    gsxr_dev_id_t                           controller_id,
    gsxr_controller_firmware_callback_fn    callback,
    void*                                   cookie);
```

说明：

通知手柄进行固件升级

参数：

[in] controller_handle 手柄设备句柄，由 open 函数获取

[in] controller_id 手柄设备识别号，当设备连接时由回调函数 gsxr_dev_event_callback_fn 获取

[in] callback 手柄固件回调函数，不可为 nullptr

[in] cookie 函数回调时带回的预备数值

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的回调函数

-4 无效的设备识别号

备注：

gsxr_dev_event_callback_fn 回调 event 为 GSXR_DEV_EVENT_CALLBACK_TYPE_DEVICE_CONNECTED，param1 即为设备识别号

[基础能力规范] gsxr_dev_controller_ops_t (*upgrade_firmware)

[参数检验]

- `controller_handle` 必须是有效的设备句柄。
- `controller_id` 必须是有效的设备识别号。
- `callback` 必须是 gsxr_controller_firmware_callback_fn 函数类型的有效指针。

[函数实现]

- XR 手柄插件记录 XR Runtime 设置的 gsxr_controller_firmware_callback_fn 函数指针，并开始 controller_id 手柄的固件升级流程。
- XR 手柄插件必须适时回调 callback 函数回报固件的更新状态及进度。

[返回值]

成功

- 0 固件升级需求设置成功。

失败

- -1 固件升级需求设置失败。
- -2 controller_handle 为无效句柄。
- -3 callback 为 nullptr。
- -4 controller_id 为无效设备识别号。

11. XR 系统函数说明

GSXR 通用设备接口将 XR 系统调控函数定义于 gsxr_dev_sys_ops_t 结构之中，除包含前面章节所描述的通用函数(详见第 5 章)外，同样也定义了 XR 系统调控的专用函数。

```
typedef struct gsxr_dev_sys_ops {
    gsxr_dev_common_ops_t common;

    int32_t (*set_thread_priority)(
                gsxr_dev_sys_handle_t sys_handle,
                uint32_t tid,
                int policy,
                int priority);
    int32_t (*set_performance_level)(
                gsxr_dev_sys_handle_t sys_handle,
                gsxr_perf_level_t cpu_perf_level,
                gsxr_perf_level_t gpu_perf_level);
```

```
    int32_t (*set_log_level)(
                gsxr_dev_sys_handle_t sys_handle,
                gsxr_log_level_t log_level);
    int32_t (*set_developer_mode)(
                gsxr_dev_sys_handle_t sys_handle,
                bool enable);
} gsxr_dev_sys_ops_t;
```

11.1. 获取 XR 系统实例

XR Runtime 在操作 XR 系统插件所实现的 gsxr_dev_sys_ops_t 成员函数之前，必须先调用 gsxr_dev_get_sys_instance 函数获取 XR 系统实例结构体 gsxr_dev_sys_t，实例结构中除可获取系统插件所实现的 gsxr_dev_sys_ops_t 所有成员函数指针外，同样可获取 GSXR 的 API 版本号，XR Runtime 必须以此实例为进入点开始 XR 系统的操作。

[结构体] gsxr_dev_sys_t (系统平台实例)

```
typedef struct gsxr_dev_sys {
    gsxr_version                    api_version;
    gsxr_dev_sys_ops_t*             sys_ops;
} gsxr_dev_sys_t;
```

说明：

系统平台实例结构体

成员：

api_version 系统实例使用的 API 版本号 GSXR_DEVICE_CURRENT_API_VERSION

sys_ops 系统实例实现的 GSXR 系统平台函数指针

[函数] gsxr_dev_get_sys_instance (获取系统平台实例)

```
gsxr_dev_sys_t* gsxr_dev_get_sys_instance();
```

说明：

获取系统平台实例

参数：

无

返回值：

系统平台实例结构体 gsxr_dev_sys_t

备注：

无

11.2. 设置线程优先级

XR Runtime 可经由 set_thread_priority 函数设置线程优先级，XR 系统插件必须具备系统权限。

[函数] gsxr_dev_sys_ops_t (*set_thread_priority) (设置线程优先级)

```
int32_t (*set_thread_priority)(
    gsxr_dev_handle_t                sys_handle,
    uint32_t                         tid,
    int                              policy,
    int                              priority);
```

说明：

设置线程优先级

参数：

[in] sys_handle 系统平台句柄，由 open 函数获取

[in] tid 线程 thread id

[in] policy 调度策略级. 设置为 SCHED_FIFO 或 SCHED_RR 可获取 real-time 级效能

[in] priority 线程优先级. 设置于 1-99 区间可获取 real-time 级效能 (数字越

高，优先级越高）

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_sys_ops_t (*set_thread_priority)

[参数检验]

- `sys_handle` 必须是有效的设备句柄。
- `tid` 必须是有效的线程识别号。
- `policy` 必须是有效的调度策略级。
- `priority` 必须是有效的线程优先级。

[函数实现]

- XR 系统插件向操作系统设置线程优先级，XR 系统插件必须具备系统权限才可以成功设置。

[返回值]

成功

- `0` 线程优先级设置成功。

失败

- `-1` 线程优先级设置失败。
- `-2` sys_handle 为无效句柄。
- `-3` tid，policy，或 priority 为无效数值。

11.3. 设置 CPU，GPU 效能等级

XR Runtime 可经由 set_performance_level 函数设置 CPU/GPU 效能等级，XR 系统插件必须具备系统权限。

[枚举] gsxr_perf_level_t (系统性能等级)

名称	数值	描述
GSXR_PERF_LEVEL_SYSTEM	0	系统预设等级
GSXR_PERF_LEVEL_HIGH	1	高性能等级
GSXR_PERF_LEVEL_MEDIUM	2	中性能等级
GSXR_PERF_LEVEL_LOW	3	低性能等级

[函数] gsxr_dev_sys_ops_t (*set_performance_level) (设置效能等级)

```
int32_t (*set_performance_level)(
    gsxr_dev_handle_t               sys_handle,
    gsxr_perf_level_t               cpu_perf_level,
    gsxr_perf_level_t               gpu_perf_level);
```

说明:

设置 CPU, GPU 效能等级

参数:

[in] sys_handle 系统平台句柄, 由 open 函数获取

[in] cpu_perf_level CPU 系统性能等级, 通常设置为 GSXR_PERF_LEVEL_SYSTEM 采取系统预设等级

[in] gpu_perf_level GPU 系统性能等级, 通常设置为 GSXR_PERF_LEVEL_SYSTEM 采取系统预设等级

返回值:

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注:

无

[基础能力规范] gsxr_dev_sys_ops_t (*set_performance_level)

[参数检验]

- sys_handle 必须是有效的设备句柄。
- cpu_perf_level 必须是有效的性能等级。
- gpu_perf_level 必须是有效的性能等级。

[函数实现]

- XR 系统插件根据设置的等级转换为系统对应的时钟频率调控 CPU/GPU，XR 系统插件必须具备系统权限才可以成功设置。

[返回值]

成功

- 0 系统效能等级设置成功。

失败

- -1 系统效能等级设置失败。
- -2 sys_handle 为无效句柄。
- -3 cpu_perf_level 或 gpu_perf_level 为无效数值。

11.4. 设置日志信息输出等级

XR Runtime 可经由 set_log_level 函数设置日志信息输出等级，XR 系统插件必须具备系统权限。

[枚举] gsxr_log_level_t (系统日志输出等级)

名称	数值	描述
GSXR_LOG_LEVEL_VERBOSE	0	输出 VERBOSE 及其以上等级的日志
GSXR_LOG_LEVEL_DEBUG	1	输出 DEBUG 及其以上等级的日志
GSXR_LOG_LEVEL_INFO	2	输出 INFO 及其以上等级的日志
GSXR_LOG_LEVEL_WARN	3	输出 WARN 及其以上等级的日志
GSXR_LOG_LEVEL_ERROR	4	只输出 ERROR 日志

[函数] gsxr_dev_sys_ops_t (*set_log_level) (设置日志信息输出等级)

```
int32_t (*set_log_level)(
    gsxr_dev_handle_t           sys_handle,
    gsxr_log_level_t            log_level);
```

说明：

设置日志信息输出等级

参数：

[in] sys_handle 系统平台句柄，由 open 函数获取

[in] log_level 系统日志输出等级

返回值：

0 成功

-1 失败

-2 无效的设备句柄

-3 无效的输入参数

备注：

无

[基础能力规范] gsxr_dev_sys_ops_t (*set_log_level)

[参数检验]

- `sys_handle` 必须是有效的设备句柄。
- `log_level` 必须是有效的日志输出等级。

[函数实现]

- XR 系统插件向操作系统设置输入的日志等级，XR 系统插件必须具备系统权限方可成功设置。

[返回值]

成功

- `0` 系统输出日志等级设置成功。

失败

- `-1` 系统输出日志等级设置失败。

- -2 sys_handle 为无效句柄。
- -3 log_level 为无效数值。

11.5. 设置开发者模式

XR Runtime 可经由 set_developer_mode 函数设置开发者模式，XR 系统插件必须具备系统权限。

[函数] gsxr_dev_sys_ops_t (*set_developer_mode) (设置开发者模式)

```
int32_t (*set_developer_mode)(
    gsxr_dev_handle_t             sys_handle,
    bool                          enable);
```

说明：

设置开发者模式

参数：

[in] sys_handle 系统平台句柄，由 open 函数获取

[in] enable 开启或关闭开发者模式，true 为开启，false 为关闭

返回值：

0 成功

-1 失败

-2 无效的设备句柄

备注：

无

[基础能力规范] gsxr_dev_sys_ops_t (*set_developer_mode)

[参数检验]

- sys_handle 必须是有效的设备句柄。

[函数实现]

- XR 系统插件根数 enable 数值开启或关闭开发者模式，XR 系统插件必须具备系统权限方可成功设置。

[返回值]

成功

- `0` 开发者模式设置成功。

失败

- `-1` 开发者模式设置失败。
- `-2` sys_handle 为无效句柄。

12. 附录

12.1. 附录一 GSXR 互通设备接口适用对象

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
Runtime 厂家	所有章节
头显设备厂家	4 XR 设备通用基础类型 5 XR 设备通用函数说明 6 XR 设备信息函数说明 7 XR 设备输入函数说明 8 XR 设备跟踪函数说明 9 XR 头戴式显示器函数说明
手柄设备厂家	4 XR 设备通用基础类型 5 XR 设备通用函数说明 6 XR 设备信息函数说明 7 XR 设备输入函数说明 8 XR 设备跟踪函数说明 10 XR 手柄控制器函数说明

12.2. 附录二 GSXR 互通设备函数索引

12.2.1. XR 设备通用函数

[gsxr_dev_common_ops_t]

- (*open) (开启设备实例)
- (*close) (关闭设备实例)
- (*set_event_callback) (设置事件回调函数)
- (*get_param) (获取设备扩展功能参数值)
- (*set_param) (设置设备扩展功能参数值)

12.2.2. XR 设备信息函数

[gsxr_dev_info_ops_t]

- (*get_dev_properties) (获取设备属性)
- (*get_battery_status) (获取设备电量状态)

12.2.3. XR 设备输入函数

[gsxr_dev_input_ops_t]

- (*get_input_click_states) (获取点击状态)
- (*get_input_touch_states) (获取触摸状态)
- (*get_input_analog_state) (获取类比数据)

12.2.4. XR 设备跟踪函数

[gsxr_dev_tracking_ops_t]

- (*get_tracking_state) (获取跟踪状态)
- (*get_tracking_mode) (获取跟踪模式)
- (*start_tracking) (开启跟踪系统)
- (*stop_tracking) (停止跟踪系统)

- (*get_pose_data)（获取跟踪姿态数据）
- (*get_sensor_data)（获取传感器数据）
- （*set_sensor_data_callback）（设置传感器数据回调函数）
- (*relocalize_origin)（重置跟踪原点）

12.2.5. XR 头显设备函数

- gsxr_dev_get_hmd_instance(获取头显设备实例)

[gsxr_dev_hmd_ops_t]
- (*get_hmd_configurations)（获取头显配置）
- (*get_hmd_ipd)（获取 IPD 距离）
- (*set_vsync_callback)（设置 V-Sync 回调函数）
- (*set_proximity_callback)（设置头部距离传感器回调函数）
- (*set_ipd_callback)（设置 IPD 回调函数）
- (*get_stream_volume)（获取头显音量状态）
- (*set_stream_volume)（设置头显音量状态）
- (*get_screen_brightness)（获取头显屏幕亮度）
- (*set_screen_brightness)（设置头显屏幕亮度）

12.2.6. XR 手柄设备函数

- gsxr_dev_get_controller_instance(获取手柄设备实例)

[gsxr_dev_controller_ops_t]
- (*start_scan_controllers)（开始搜索手柄）
- (*get_available_controllers)（获取搜索到的手柄列表）
- (*stop_scan_controllers)（停止搜索手柄）
- (*connect)（连接手柄）
- (*disconnect)（断开手柄）
- (*get_paired_controllers)（获取曾配对过的手柄列表）
- (*get_controller_configurations)（获取手柄配置）
- (*get_touchpad_mode)（获取触控模式）
- (*enable_touchpad)（初始化触控板）

- (*set_touchpad_mode)（设置触控模式）
- (*disable_touchpad)（禁用触控板）
- (*trigger_vibration)（触发振动反馈）
- (*upgrade_firmware)（升级固件）

12.2.7. XR 系统函数

- gsxr_dev_get_sys_instance(获取系统平台实例)

[gsxr_dev_sys_ops_t]
- (*set_thread_priority)（设置线程优先级）
- (*set_performance_level)（设置效能等级）
- (*set_log_level)（设置日志信息输出等级）
- (*set_developer_mode)（设置开发者模式）

中国 VRPC 联盟标准

GSXR 互通接口与基础能力规范

GSXR-D-01-04