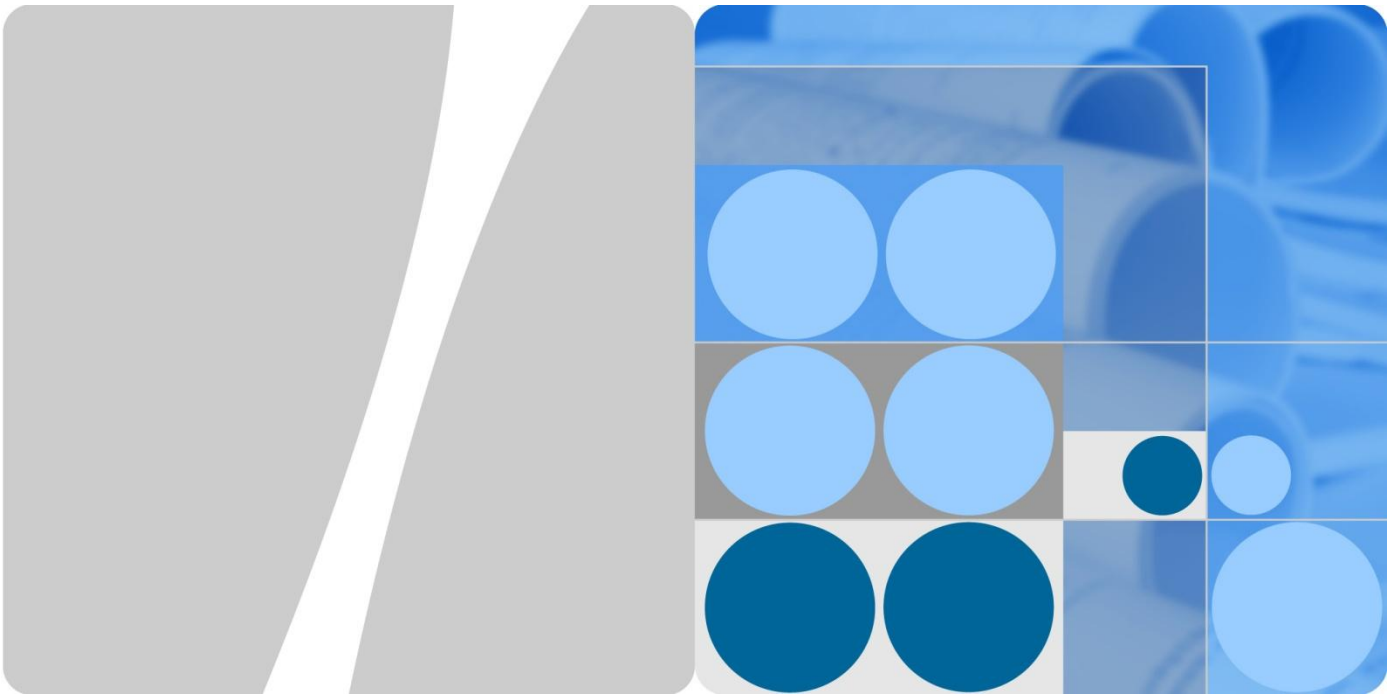


HuaweiVR SDK2.0 Unity API Description

Issue 2.0.0

Date 2017-09-08

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions

and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.huawei.com

Email: support@huawei.com

Change History

Date	Version	Description
2017-09-08	2.0.0	This is the second edition.

Contents

Change History ii

1 Description of Variables and Interface Functions 1

2 Public Enumerations 2

2.1 TrackStatus 3

2.2 HelmetModel 3

2.3 ControllerType 4

2.4 ControllerEvent 4

2.5 ControllerStatus 4

2.6 ButtonType 5

3 Public Structures 7

3.1 Posture 7

3.2 SensorData 8

3.3 RenderStatistics 8

4 Public Classes 10

4.1 HVRLayoutCore 10

4.1.1 m_CamCtrObj 10

4.1.2 m_LeftCamObj 11

4.1.3 m_RightCamObj 11

4.2 HVRCamCore 11

4.2.1 UseSkyBox 11

4.3 HVRDefCore 12

4.3.1 SafeModeColor 12

4.4 HvrApi 13

4.4.1 GetHvrSdkVersion 13

4.4.2 GetHelmetHandle 13

4.4.3 GetControllerHandle 14

4.4.4 GetRenderHandle 14

4.4.5 EnableSvsEffect 15

4.5 IHelmetHandle 15

4.5.1 IsAvailable 16

4.5.2 GetPosture 17

4.5.3 GetSensorInfo 18
4.5.4 ResetCenter 19
4.5.5 ResetYaw 19
4.5.6 ResetOrientation 20
4.5.7 ResetPosition (Not Supported) 20
4.5.8 SetPoseLock 20
4.5.9 SetPositionLock (Not Supported) 21
4.5.10 GetHelmetInfo 22
4.6 IControllerHandle 23
4.6.1 ControllerEventArgs 23
4.6.2 ControllerStatusEventHandler 23
4.6.3 GetValidIndices 24
4.6.4 GetControllerByIndex 25
4.6.5 IsLeftHandMode 26
4.7 IController 26
4.7.1 IsAvailable 28
4.7.2 GetControllerStatus 28
4.7.3 ResetCenter(Not Supported) 29
4.7.4 GetControllerType (Not Supported) 29
4.7.5 GetBatteryLevel (Not Supported) 30
4.7.6 GetPosture 30
4.7.7 GetTriggerData 31
4.7.8 GetTouchpadTouchPos 32
4.7.9 GetGyroscope (Not Supported) 34
4.7.10 GetAccelerometer (Not Supported) 34
4.7.11 IsTouchpadTouching 35
4.7.12 IsTouchpadTouchDown 35
4.7.13 IsTouchpadTouchUp 35
4.7.14 IsTouchpadSwipeUp 36
4.7.15 IsTouchpadSwipeDown 36
4.7.16 IsTouchpadSwipeLeft 36
4.7.17 IsTouchpadSwipeRight 36
4.7.18 IsButtonPressed 37
4.7.19 IsButtonDown 38
4.7.20 IsButtonUp 38
4.7.21 StartVibration() (Not Supported) 39
4.7.22 StopVibration() (Not Supported) 39
4.8 IRenderHandle 40
4.8.1 CaptureEyeImage 40
4.8.2 EnableChromaticAberration 41
4.8.3 GetRenderStatics 42
4.8.4 EnableMonocularDisplay 43

4.9 HVRPluginCore 43
4.9.1 HVR_SetClipPlaneParams 44
4.9.2 HVR_OpenSafeMode 44
4.9.3 HVR_CloseSafeMode 45
4.9.4 HVR_SetSafeModeParams 45
4.10 HvrEventListner 46

1 Description of Variables and Interface Functions

Variables and interface functions are all compiled using the C# language and saved as a **.**dll file. Before using these variables and functions, the HVRCORE namespace needs to be introduced. That is, add the **using HVRCORE** statement. The HVRCORE namespace provides public enumerations, public structures, and public classes of HuaweiVR SDK2.0 for Unity (SDK2.0 for short).

2 Public Enumerations

Public Enumeration	

enum TrackStatus {
Tracking=0,
Untracking=1
}
	Indicates the helmet tracking status.

enum HelmetModel {
HVR_HELMET_FIRST_GEN,
HVR_HELMET_SECOND_GEN,
HVR_HELMET_NOT_FOUND,
HVR_HELMET_UNKNOWN
}
	Indicates the helmet type.

enum ControllerType {
Controller3DOF,
Controller6DOF,
ControllerGaming,
ControllerSysStd,
ControllerOther
}
	Indicates the controller type.

enum ControllerEvent {
ControllerEventConnected,
ControllerEventDisconnected,
ControllerEventLowPower,
}
	Indicates the controller connection event.

enum ControllerStatus{ ControllerStatusDisconnected, ControllerStatusScanning, ControllerStatusConnecting, ControllerStatusConnected,	Indicates the controller connection status.

Public Enumeration	
ControllerStatusError }	

enum ButtonType {
ButtonHome,
ButtonBack,
ButtonVolumeInc,
ButtonVolumeDec,
ButtonConfirm,
ButtonTrigger,
ButtonTouchPad
}
	Indicates the controller button type.

2.1 TrackStatus

TrackStatus

TrackStatus indicates whether trace data of the helmet is valid or not.

Attribute	
Tracking	Indicates that trace data of the helmet is valid.
Untracking	Indicates that trace data of the helmet is invalid.

2.2 HelmetModel

HelmetModel

HelmetModel indicates the type of a HuaweiVR helmet.

Attribute	
HVR_HELMET_FIRST_GEN	Indicates a HuaweiVR1.0 helmet.
HVR_HELMET_SECOND_GEN	Indicates a HuaweiVR2.0 helmet.
HVR_HELMET_NOT_FOUND	Indicates that no helmet is found.
HVR_HELMET_UNKNOWN	Indicates an unknown helmet type.

2.3 ControllerType

ControllerType

ControllerType indicates the type of a controller. It is not supported currently.

Attribute	
Controller3DOF	Indicates a 3Dof controller.
Controller6DOF	Indicates a 6Dof controller.
ControllerGaming	Indicates a gaming-based controller.
ControllerSysStd	Indicates a controller of standard system input type.
ControllerOther	Indicates a controller of other type.

2.4 ControllerEvent

ControllerEvent

ControllerEvent indicates the type of a controller connection event. Controller status change events can be monitored by registering controller event handles.

Attribute	
ControllerEventConnected	Indicates that a controller is connected.
ControllerEventDisconnected	Indicates that a controller is disconnected.
ControllerEventLowPower	Indicates a low-power alarm event.

2.5 ControllerStatus

ControllerStatus

ControllerStatus indicates the controller connection status, which can be actively queried.

Attribute	
ControllerStatusDisconnected	Indicates that a controller is disconnected.
ControllerStatusScanning	Indicates that a controller is scanning.

Attribute	
ControllerStatusConnecting	Indicates that a controller is connecting.
ControllerStatusConnected	Indicates that a controller is connected.
ControllerStatusError	Indicates the controller connection error.

2.6 ButtonType

ButtonType

ButtonType indicates the types of all buttons on a controller.

Attribute	
ButtonHome	Indicates the home button of the controller. It is not reported.
ButtonBack	Indicates the back button of the controller.
ButtonVolumeInc	Indicates the volume-up button of the controller. It is not reported.
ButtonVolumeDec	Indicates the volume-down button of the controller. It is not reported.
ButtonConfirm	Indicates the confirm button of the controller.
ButtonTrigger	Indicates the trigger button of the controller.
ButtonTouchPad	Indicates the touchpad of the controller.

SDK2.0 supports the shortcut and compound buttons of controllers as well.

Operation	Button	Button Holding Duration	Function
Click	Home	< 1s	Navigate to the **VRLauncher** page.
Long press	Home	≥ 1s	Reset the viewpoint.
Long press	Back	≥ 3s	Navigate to the **VRSetting** page.
Long press	Home and Trigger	≥ 1s	Capture a screenshot.

NOTE

- The shortcut and compound buttons of controllers can be used only after applications can call the controller functions.
- Before using the screenshot capturing function, set the write permission on the Unity platform by choosing **Player Settings…** > **Other Settings** > **Write Permission** and selecting **External (SDCard)**.

3 Public Structures

Public Structure	

struct Posture{
public Quaternion rotation;
public Vector3 position;
public TrackStatus trackStatus;
}
	Indicates the posture, position, and tracking status.

struct SensorData{
public Vector3 gypo;
public Vector3 accel;
public Vector3 mag;
public ulong predictTime;
}
	Indicates the helmet sensor data.

struct RenderStatistics{
public double SubmitFrameRate;
public double RenderFrameRate;
public double ATWFrameCnt;
public double AvgRenderCost;
public double TimeoutFrameCnt;
public double ContinousTimeoutCnt;
}
	Indicates the statistical parameters.

3.1 Posture

Posture

Posture indicates the posture, position, and tracking status and is a structure consisting of the rotation quaternion, position vector, and tracking status of the helmet or controller.

Attribute		
Member	**Type**	**Description**
rotation	Quaternion	Indicates the rotation quaternion (x, y, z, w).
position	Vector3	Indicates the position vector (x, y, z).
trackStatus	TrackStatus	Indicates the head tracking status.

3.2 SensorData

SensorData

SensorData indicates helmet sensor data and is a structure consisting of the gyro data, accelerometer data, magnetometer data, and prediction time of the helmet.

Attribute		
Member	**Type**	**Description**
gypo	Vector3	Indicates the gyro data.
accel	Vector3	Indicates the accelerometer data.
mag	Vector3	Indicates the magnetometer data.
predictTime	ulong	Indicates the prediction time (unit: us).

3.3 RenderStatistics

RenderStatistics

RenderStatistics indicates the rendering information statistics and is a structure consisting of the frame submit rate, frame rendering rate, number of ATW rendering frames, number of rendering timeout frames, and number of frames with continuous rendering timeout.

Attribute		
Member	**Type**	**Description**
SubmitFrameRate	double	Indicates the frame submit rate.
RenderFrameRate	double	Indicates the frame rendering rate.

Attribute		
Member	**Type**	**Description**
ATWFrameCnt	double	Indicates the number of ATW rendering frames.
AvgRenderCost	double	Indicates the average rendering time consumed per frame.
TimeoutFrameCnt	double	Indicates the total number of rendering timeout frames.
ContinousTimeoutCnt	double	Indicates the number of frames with continuous rendering timeout.

4 Public Classes

4.1 HVRLayoutCore

The HVRLayout class provides camera components for SDK2.0.

Overview

This class includes camera, left-view, and right-view transform components.

This class inherits from the MonoBehaviour class.

Public Static Attributes

Attribute Name	Type	Description
m_CamCtrObj	Transform	Indicates the camera transform component.
m_LeftCamObj	Transform	Indicates the left-view transform component.
m_RightCamObj	Transform	Indicates the right-view transform component.

4.1.1 m_CamCtrObj

Transform m_CamCtrObj

The camera transform component is used to obtain information about the position, rotation, scaling of a HuaweiVR camera.

Examples

```
HVRLayoutCore.m_CamCtrObj.transform.position; //This parameter is used to obtain camera position information.
HVRLayoutCore.m_CamCtrObj.transform.rotation; //This parameter is used to obtain camera rotation information.
```

4.1.2 m_LeftCamObj

Transform m_LeftCamObj

The left-view transform component is used to obtain information about the position, rotation, and scaling of a left view.

4.1.3 m_RightCamObj

Transform m_RightCamObj

The right-view transform component is used to obtain information about the position, rotation, and scaling of a right view.

4.2 HVRCamCore

The HVRCamCore class provides the skybox function.

Overview

This class provides the skybox function to customize application scenes by replacing skybox materials.

This class inherits from the MonoBehaviour class.

Public Static Methods

Method Name	Return Value Type	Description
UseSkyBox(bool bUse, Material mat = null)	bool	Set an application scene by replacing skybox materials.

4.2.1 UseSkyBox

The **bool UseSkyBox(bool bUse, Material mat = null)** function indicates whether the skybox is used. If it is used, skybox materials can be replaced to set an application scene.

Parameters

Parameter Name	Parameter Type	Description
bUse	bool	If the value is **true**, a skybox is drawn. If the value is **false**, no skybox is drawn.
mat	Material	Indicates the skybox material.

Return Values

During skybox drawing, **true** is returned after the skybox material is successfully replaced and **false** is returned after the skybox material fails to be replaced. If no skybox is drawn, **true** is returned.

Examples

```
Material skyboxmat = Resources.Load ("Materials/Skybox") as Material;
if (skyboxmat != null) {
    bool ret = HVRCamCore.UseSkyBox (true, skyboxmat);
    if (ret){
        Debug.Log ("Materials load success!");
    }else{
        Debug.Log ("Materials load failed!");
    }
} else {
    Debug.Log ("material not loaded");
}
```

4.3 HVRDefCore

The HVRDefCore class provides public enumerations for image colors in the safe mode.

Overview

This class provides the types of image colors after the safe mode is set.

4.3.1 SafeModeColor

SafeModeColor indicates the types of image colors after the safe mode is set.

Attribute	
Default	Indicates the default color.
White	Indicates the white color.
Red	Indicates the red color.
Green	Indicates the green color.
Blue	Indicates the blue color.
Yellow	Indicates the yellow color.
Cyan	Indicates the cyan color.
Magenta	Indicates the magenta color.

4.4 HvrApi

This class is the main entry point for invoking some APIs in SDK2.0 Unity.

Overview

This class is used to obtain SDK2.0 version numbers and handles of each module.

Public Static Methods

Method Name	Return Value Type	Description
GetHvrSdkVersion()	string	Is used to obtain the SDK2.0 version number.
GetHelmetHandle ()	IHelmetHandle	Is used to obtain the helmet handle.
GetControllerHandle ()	IControllerHandle	Is used to obtain the controller handle.
GetRenderHandle()	IRenderHandle	Is used to obtain the render handle.

4.4.1 GetHvrSdkVersion

The **string GetHvrSdkVersion()** function is used to obtain SDK2.0 version numbers.

Return Values

If the function is successfully called, the character string of the SDK2.0 version number is returned. If the function fails to be called, an empty character string is returned.

Examples

```
string version = HvrApi.GetHvrSdkVersion ();
Debug.Log ("SDK2.0 version:"+ version);
```

4.4.2 GetHelmetHandle

The **IHelmetHandle GetHelmetHandle ()** function is used to obtain the helmet handle (IHelmetHandle) and to call helmet-related interface functions. For details, see section 4.5 "IHelmetHandle."

Return Values

If the function is successfully called, **IHelmetHandle** is returned. If the function fails to be called, **null** is returned.

NOTE

This function can be called only in Start() or subsequent functions of the Unity life cycle.

Examples

```
IHelmetHandle helmetHandle; //This variable related to the helmet handle is used in the
//following codes.
helmetHandle = HvrApi.GetHelmetHandle ();
if (helmetHandle != null) {
    Debug.Log (" GetHelmetHandle Success! ");
} else {
    Debug.Log (" GetHelmetHandle Failed! ");
}
```

4.4.3 GetControllerHandle

The **IControllerHandle GetControllerHandle ()** function is used to obtain the controller handle **IControllerHandle** and to call controller-related interface functions. For details, see section 4.6 "IControllerHandle."

Return Values

If the function is successfully called, **IControllerHandle** is returned. If the function fails to be called, **null** is returned.

NOTE

This function can be called only in Start() or subsequent functions of the Unity life cycle.

Before this function is called in a non-script thread or application main thread, the current thread needs to be bound with the Java thread. If this function does not need to be called, the current thread needs to be unbound with the Java thread.

Examples

```
IControllerHandle controllerHandle; //This variable related to the controller is used in the
following codes.
controllerHandle = HvrApi.GetControllerHandle ();
if (controllerHandle != null) {
    Debug.Log (" GetControllerHandle Success! ");
} else {
    Debug.Log (" GetControllerHandle Failed! ");
}
```

4.4.4 GetRenderHandle

This **IRenderHandle GetRenderHandle()** function is used to obtain the render handle and to call render-related interface functions. For details, see section 4.8 "IRenderHandle."

Return Values

If the function is successfully called, **IRenderHandle** is returned. If the function fails to be called, **null** is returned.

NOTE

This function can be called only in Start() or subsequent functions of the Unity life cycle.

Examples

```
IRenderHandle renderHandle; //This variable related to the rendering handle is used in the following codes.
renderHandle = HvrApi.GetRenderHandle ();
if (renderHandle != null) {
    Debug.Log ("GetRenderHandle Success!");
} else {
    Debug.Log ("GetRenderHandle Failed!");
}
```

4.4.5 EnableSvsEffect

The **int EnableSvsEffect (bool enable)** function is used to enable the SVS audio effect of the EMUI system. This function requires the **android.permission.MODIFY_AUDIO_SETTINGS** permission.

After the SVS audio effect is enabled, the positions of all sounds in an application change with the head direction. This function applies to video watching in a VR. You need to pay attention to the enabling time and disable the function in a timely manner.

Return Values

0 is fixedly returned.

Examples

```
HvrApi.EnableSvsEffect (true);
```

4.5 IHelmetHandle

The IHelmetHandle class provides helmet-related functions.

Overview

This class provides the functions of obtaining the posture, position and sensor data, locking and unlocking postures.

Public Methods

Method Name	Return Value Type	Description
IsAvailable()	bool	Is used to query whether a device is available.
GetPosture(ref Posture posture)	int	Is used to obtain the position and posture of the current helmet.
GetSensorInfo (ref SensorData sensor)	int	Is used to obtain the sensor data and prediction time of the helmet.
ResetCenter()	int	Is used to reset the screen position and posture to the initial status.
ResetYaw ()	int	Is used to reset the screen yaw angle.
ResetOrientation ()	int	Is used to reset the gyro posture.
ResetPosition ()	int	Is used to reset the current position to the original.
SetPoseLock (bool enable)	int	Is used to enable or disable posture lock.
SetPositionLock (bool enable)	int	Is used to enable or disable position lock.
GetHelmetInfo(ref HelmetModel helmetModel)	int	Is used to obtain the helmet type.

4.5.1 IsAvailable

The **bool IsAvailable()** function is used to query whether a helmet device is available or not. In most cases, this function is called after the helmet handle is obtained.

Return Values

If the device is available, **true** is returned. If the device is unavailable, **false** is returned.

Examples

```
if (helmetHandle != null) {
    Debug.Log (" GetHelmetHandle Success! ");
    if (helmetHandle.IsAvailable ()) {
        Debug.Log (" Helmet is available! ");
    } else {
        Debug.Log (" Helmet is not available! ");
    }
} else {
    Debug.Log (" GetHelmetHandle Failed! ");
}
```

4.5.2 GetPosture

The **int GetPosture(ref Posture posture)** function is used to obtain the position vector, tracking status, and rotation quaternion of the current helmet.

Parameters

Parameter Name	Parameter Type	Description
posture	Posture For details, see section 4.1 "HVRLayoutCore."	Writes the rotation quaternion (x, y, z, w) of the current helmet into **posture.rotation**.
		Writes the position vector (x, y, z) of the current helmet into **posture.position**.
		Writes the tracking status (tracking or untracking) of the current helmet into **posture.trackStatus**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
Posture pos = new Posture ();
int ret = helmetHandle.GetPosture (ref pos);
if (ret == 0) {
    Quaternion quatDat;
    Vector3 posDat;
    quatDat = pos.rotation; //This parameter is used to obtain the rotation posture
//information.
    posDat = pos.position; //This parameter is used to obtain the position information.
} else {
    Debug.Log (" Get VR glass posture failed! ");
}
```

4.5.3 GetSensorInfo

The **int GetSensorInfo (ref SensorData sensor)** function is used to obtain the sensor data and prediction time of the current helmet.

Parameters

Parameter Name	Parameter Type	Description
sensor	SensorData For details, see section 3.2 "SensorData."	Writes the gyro data (x, y, z) of the helmet into **sensor.gypo**.
		Writes the accelerometer data (x, y, z) of the helmet into **sensor.accel**.
		Writes the magnetometer data (x, y, z) of the helmet into **sensor.mag**
		Writes the prediction time (unit: us) of the helmet into **sensor.predictTime**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
SensorData sensor = new SensorData();
int ret = helmetHandle.GetSensorInfo (ref sensor);
if( ret == 0 ){
    Vector3 gypoDat;
    gypoDat = sensor.gypo; //This parameter is used to obtain the gyro data of the helmet.
}else{
    Debug.Log(" Get VR glass SensorData failed! ");
}
```

4.5.4 ResetCenter

The **int ResetCenter()** function is used to reset the current position and posture to the initial status. It applies when the screen posture and position deviate. However, position reset is not supported currently.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
//Reset the position and posture.
if(helmetHandle.ResetCenter () == 0 ){
    Debug.Log(" Reset Center Success! ");
}else{
    Debug.Log(" Reset Center Failed! ");
}
```

4.5.5 ResetYaw

The **int ResetYaw ()** function is used to reset the yaw angle. In most cases, when users take off the VR helmet, the smartphone screen goes out. When users wear the helmet gain and the screen goes on, the screen view angle may deviate.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
//Reset the yaw angle.
if(helmetHandle.ResetYaw () == 0 ){
    Debug.Log(" Reset Yaw Success! ");
}else{
    Debug.Log(" Reset Yaw Failed! ");
}
```

4.5.6 ResetOrientation

The **int ResetOrientation ()** function is used to reset the gyro sensor posture to the original one. It applies when the screen posture deviates.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
//Reset the posture.
if(helmetHandle.ResetOrientation () == 0 ){
    Debug.Log(" Reset orientation Success! ");
}else{
    Debug.Log(" Reset orientation Failed! ");
}
```

4.5.7 ResetPosition (Not Supported)

The **int ResetPosition ()** function is used to reset the current position to the original one. It applies when the screen position deviates.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

4.5.8 SetPoseLock

The **int SetPoseLock (bool enable)** function is used to enable or disable posture lock. It is used during movie watching. This function is called to enable or disable the posture lock before and after a viewer adjusts the viewing posture, thereby ensuring that the screen posture does not change.

Parameter

Parameter Name	Parameter Type	Description
enable	bool	If the value is **true**, the posture lock is enabled. If the value is **false**, the posture lock is disabled.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(helmetHandle.SetPoseLock (true) == 0 ){
    Debug.Log(" Set Pose Lock Success! " );
}else{
    Debug.Log(" Set Pose Lock Failed ! ");
}
if(helmetHandle.SetPoseLock (false) == 0 ){
    Debug.Log(" Set Pose Unlock Success ! " );
}else{
    Debug.Log(" Set Pose Unlock Failed! ");
}
```

4.5.9 SetPositionLock (Not Supported)

The **int SetPositionLock (bool enable)** function is used to enable or disable position lock. It is used during movie watching. This function is called to enable or disable the position lock before and after a viewer adjusts the viewing position, thereby ensuring that the screen position does not deviate.

Parameter

Parameter Name	Parameter Type	Description
enable	bool	If the value is **true**, the position lock is enabled. If the value is **false**, the position lock is disabled.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

4.5.10 GetHelmetInfo

The **int GetHelmetInfo(ref HelmetModel helmetModel)** function is used to obtain the helmet type, such as HuaweiVR1.0, HuaweiVR2.0, or others.

Parameter

Parameter Name	Parameter Type	Description
helmetModel	HelmetModel For details, see section 2.2 "HelmetModel."	Writes the helmet type (HVR_HELMET_FIRST_GEN, HVR_HELMET_SECOND_GEN, HVR_HELMET_NOT_FOUND, or HVR_HELMET_UNKNOWN) into **helmetModel**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
HelmetModel helmetmodel = HelmetModel.HVR_HELMET_UNKNOWN;
int ret = helmetHandle.GetHelmetInfo(ref helmetmodel);
if( ret == 0 ){
    switch(helmetmodel){
    case HelmetModel.HVR_HELMET_FIRST_GEN:
        Debug.Log("HuaweiVR1.0 helmet " );
        break;
    case HelmetModel.HVR_HELMET_SECOND_GEN:
        Debug.Log("HuaweiVR2.0 helmet " );
        break;
    case HelmetModel.HVR_HELMET_NOT_FOUND:
        Debug.Log(" not found helmet " );
        break;
    case HelmetModel.HVR_HELMET_UNKNOWN:
        Debug.Log(" unknown helmet " );
        break;
    }
}
```

4.6 IControllerHandle

The IControllerHandle class monitors controller events and event handles and obtains controller objects and serial numbers of available controllers.

Overview

This class monitors controller events and obtains controller objects and serial numbers of available controllers.

Event

Event Name	Inherited From	Description
ControllerEventArgs	EventArgs	Indicates the event reported by a controller.

4.6.1 ControllerEventArgs

This function monitors controller disconnection, connection, and low-power events. It contains the following two attributes.

Attribute Name	Type	Description
eventId	ControllerEvent	Indicates the controller event ID, which identifies a controller disconnection, connection, or low-power event. Currently, the low-power event is not supported. For details, see section 2.4 "ControllerEvent."
eventData	Object	Indicates the controller event data, which identifies the serial number of a connected or disconnected controller.

Event Handle

Handle Name	Type	Description
ControllerStatusEventHandler	event EventHandler	Indicates the controller event handle.

4.6.2 ControllerStatusEventHandler

The **event EventHandler ControllerStatusEventHandler** function indicates the controller event handle. A controller handle is registered to monitor controller disconnection, connection, and low-power events.

Examples

```
void OnEnable ()
{
    controllerHandle.ControllerStatusEventHandler += Handle_Controller;
}
void Handle_Controller (object sender, EventArgs args)
{
    var controllerArgs = (ControllerEventArgs)args;
    ControllerEvent controllerEvent = (EventArgs)controllerArgs.eventId;
    int controllerIndex = (int)controllerArgs.eventData;
    switch (controllerEvent) {
    case ControllerEvent.ControllerEventConnected:
            Debug.Log (" Controller is connected " + controllerIndex);
            break;
    case ControllerEvent.ControllerEventDisconnected:
            Debug.Log (" Controller is disconnected " + controllerIndex);
            break;
    case ControllerEvent.ControllerEventLowPower:
            Debug.Log (" Controller is low power ");
            break;
    }
}
```

Public Methods

Method Name	Return Value Type	Description
GetValidIndices ()	int[]	Is used to obtain the serial numbers of available controllers.
GetControllerByIndex (int index)	IController	Is used to obtain the controller handle whose serial number is index.
IsLeftHandMode()	bool	Checks whether the controller is in the left-hand mode.

4.6.3 GetValidIndices

The **int[] GetValidIndices ()** function is used to obtain the serial numbers of available controllers which are not all connected controllers. When the serial number is 0, a standard input event is used for HuaweiVR1.0 helmet touchpad. If the serial number is 1, 2, 3, …, a controller input event is used.

Return Values

If the function is successfully called, the serial number array of an available controller is returned. If the function fails to be called, **null** is returned.

Examples

```
if(null == controllerHandle){
    Debug.LogError("controllerHandle is null ");
}else{
    int[] indices = controllerHandle.GetValidIndices ();
}
```

4.6.4 GetControllerByIndex

The **IController GetControllerByIndex(int index)** function is used to obtain the controller object whose serial number is index.

Parameter

Parameter Name	Parameter Type	Description
index	int	Indicates the serial number of a controller.

Return Values

If the function is successfully called, the controller object whose serial number is index is returned. If the function fails to be called, **null** is returned.

Examples

```
private IController controller; //This variable related to the controller is used in the following codes.
void Start(){
    if(null == controllerHandle){
        Debug.LogError("controllerHandle is null ");
    }else{
        int[] indices = controllerHandle.GetValidIndices ();
        //Obtain the controller object of the controller input event.
        controller = controllerHandle.GetControllerByIndex(indices[1]);
    }
    if(null == controller){
        Debug.LogError(" Controller is null ");
    }
```

```
}
```

4.6.5 IsLeftHandMode

The **bool IsLeftHandMode()** function indicates whether left-hand mode is used. Developers need to call this function to adjust the virtual controller display position when the **OnApplicationPause** value is **false**.

Return Values

If the function is successfully called and **true** is returned, the left-hand mode is used. If **false** is returned, the left-hand mode is not used.

Examples

```
void OnApplicationPause(bool isPause){
    if( !isPause){
        if(null == controllerHandle){
            Debug.LogError("controllerHandle is null ");
        }else {
            if(controllerHandle.IsLeftHandMode()){
                  Debug.LogError("controller is in leftHand mode! ");
             }
        }
      }
 }
```

4.7 IController

The IController class provides controller-related functions.

Overview

This class supports the following functions:

- Obtain the status, type, power, posture, position of a controller.
- Acquire the trigger data, touch position of the touchpad, gyro data, and accelerometer data.
- Reset the controller posture.
- Determine push-button actions of a touchpad on a controller.
- Set controller vibration.

Public Methods

Method Name	Return Value Type	Description
IsAvailable()	bool	Is used to query whether a controller device is available.
GetControllerStatus()	ControllerStatus	Is used to obtain the status of a controller.
ResetCenter()	int	Is used to reset the current position and posture to the initial status.
GetControllerType()	ControllerType	Is used to obtain the controller type.
GetBatteryLevel()	int	Is used to obtain the controller power.
GetPosture(ref Posture pose)	int	Is used to obtain the controller position and posture.
GetTriggerData(ref float data)	int	Is used to obtain the controller trigger data.
GetTouchpadTouchPos (ref Vector2 pos)	int	Is used to obtain the touch position of a touchpad on a controller.
GetGyroscope(ref Vector3 gyroscope)	int	Is used to obtain the controller gyro data.
GetAccelerometer (ref Vector3 accelerometer)	int	Is used to obtain the controller accelerometer data.
IsTouchpadTouching()	bool	Determines whether a current touchpad is touched.
IsTouchpadTouchDown()	bool	Determines whether a touch-down action is triggered on the touchpad.
IsTouchpadTouchUp()	bool	Determines whether a touch-up action is triggered on the touchpad.
IsTouchpadSwipeUp()	bool	Determines whether a swipe-up action is triggered on the touchpad.
IsTouchpadSwipeDown()	bool	Determines whether a swipe-down action is triggered on the touchpad.
IsTouchpadSwipeLeft()	bool	Determines whether a left-swipe action is triggered on the touchpad.
IsTouchpadSwipeRight()	bool	Determines whether a right-swipe action is triggered on the touchpad.
IsButtonPressed(ButtonType button)	bool	Determines whether the controller button is pressed.
IsButtonDown(ButtonType button)	bool	Determines whether a touch-down action is triggered on the controller button.

Method Name	Return Value Type	Description
IsButtonUp(ButtonType button)	bool	Determines whether a touch-up action is triggered on the controller button.
StartVibration()	int	Enables the controller to start vibration.
StopVibration()	int	Enables the controller to stop vibration.

4.7.1 IsAvailable

The **bool IsAvailable ()** function is used to query whether a connected controller is available.

Return Values

If **true** is returned, the controller is available. If **false** is returned, the controller is unavailable.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    if(controller.IsAvailable ()){
        Debug.Log (" Controller is available ");
    }
}
```

4.7.2 GetControllerStatus

The **ControllerStatus GetControllerStatus()** function is used to obtain the status of a controller.

Return Values

The current connection status of a controller is returned. For details, see section 2.5 "ControllerStatus."

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    ControllerStatus status = controller. GetControllerStatus();
switch (status) {
case ControllerStatus.ControllerStatusDisconnected:
```

```
    Debug.Log (" Controller is disconnected ");
    break;
case ControllerStatus.ControllerStatusScanning:
    Debug.Log (" Controller is scanning ");
    break;
case ControllerStatus.ControllerStatusConnecting:
    Debug.Log (" Controller is connecting ");
    break;
case ControllerStatus.ControllerStatusConnected:
    Debug.Log (" Controller is connected ");
    break;
case ControllerStatus.ControllerStatusError:
    Debug.Log (" Controller is connect error ");
    break;
}}
```

4.7.3 ResetCenter(Not Supported)

The **int ResetCenter()** function is used to reset the current position and posture to the initial status. It applies when the screen posture and position deviate. However, position reset is not supported currently.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    controller. ResetCenter();
}
```

4.7.4 GetControllerType (Not Supported)

The **ControllerType GetControllerType()** function is used to obtain the controller types, such as 3Dof, 6Dof, gaming-based, or standard system input.

Return Values

If this function is successfully called, the type of the currently connected controller is returned. For details, see section 2.3 "ControllerType."

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    controllerType type = Controller.GetControllerType ();
}
```

4.7.5 GetBatteryLevel (Not Supported)

The **int GetBatteryLevel()** function is used to obtain the power value of a controller.

Return Values

If the function is successfully called, the power value of the controller is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    int batteryLevel = controller.GetBatteryLevel();
}
```

4.7.6 GetPosture

The **int GetPosture(ref Posture pose)** function is used to obtain the position vector, tracking status, and rotation quaternion of the current controller. However, obtaining the position vector is not supported.

Parameter

Parameter Name	Parameter Type	Description
pose	Posture For details, see section 3.1 "Posture."	Writes the rotation quaternion (x, y, z, w) of the current controller into **pose.rotation**.
		Writes the position vector (x, y, z) of the current controller into **pose.position**.
		Writes the tracking status (tracking or untracking) of the current controller into **pose.trackStatus**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    Posture pos = new Posture ();
    int ret = controller.GetPosture(ref pos);
    if( ret == 0 ){
        Quaternion quatDat;
        Vector3 posDat;
        quatDat = pos.rotation; //Obtain the rotation posture of a controller.
        posDat = pos.position; //Obtain the controller position.
    }else{
        Debug.Log(" Get controller posture failed! ");
    }
}
```

4.7.7 GetTriggerData

The **int GetTriggerData(ref float data)** function is used to obtain the trigger data of the current controller.

Parameter

Parameter Name	Parameter Type	Description
data	float	Writes the trigger data (value: floating point numbers of [0.0f, 1.0f]) into **data**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    float triggerData = 0.0f;
    int ret = controller.GetTriggerData(ref triggerData);
    if( ret == 0 ){
        Debug.Log(" Get controller trigger data success! ");
    }else{
        Debug.Log(" Get controller trigger data failed! ");
    }
}
```

NOTE

The trigger value of controllers is **0** or **1** in HuaweiVR SDK2.0.

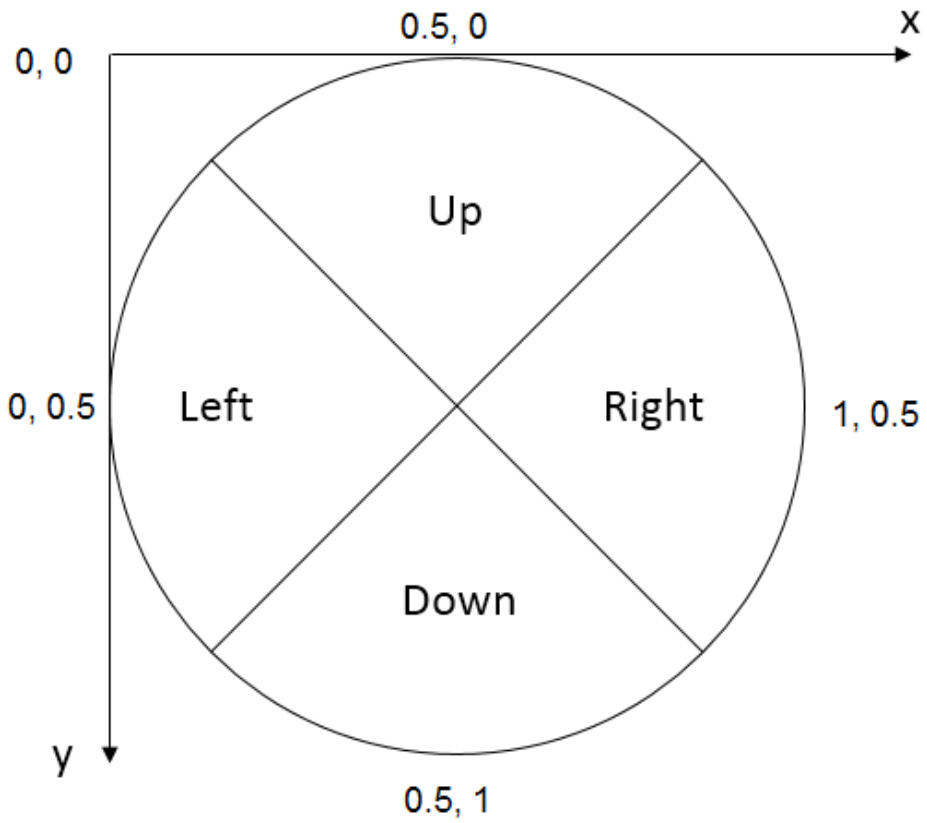
4.7.8 GetTouchpadTouchPos

The **int GetTouchpadTouchPos(ref Vector2 pos)** function is used to obtain the touch position vector of a touchpad of the current controller.

Parameter

Parameter Name	Parameter Type	Description
pos	Vector2	Writes the position vector (x, y) of the current controller into **pos**.

The following figure shows the coordinate system of the obtained position vector from the controller touchpad. When the position vector is used, the data needs to be converted by the actual coordinate.



Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    Vector2 touchpdPos = new Vector2();
    int ret = controller.GetTouchpadTouchPos(ref touchpdPos);
    if( ret == 0 ){
        Debug.Log(" Get controller touchpdPos success! ");
    }else{
        Debug.Log(" Get controller touchpdPos failed! ");
    }
}
```

4.7.9 GetGyroscope (Not Supported)

The **int GetGyroscope(ref Vector3 gyroscope)** function is used to obtain the gyro data of the current controller.

Parameter

Parameter Name	Parameter Type	Description
gyroscope	Vector3	Writes the gyro position vector (x, y, z) of the current controller into **gyroscope**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    Vector3 gyroscope = new Vector3();
    int ret = controller.GetGyroscope(ref gyroscope);
    if( ret == 0 ){
        Debug.Log(" Get controller gyroscope success! ");
    }else{
        Debug.Log(" Get controller gyroscope failed! ");
    }
}
```

4.7.10 GetAccelerometer (Not Supported)

The **int GetAccelerometer(ref Vector3 accelerometer)** function is used to obtain the accelerometer data of the current controller.

Parameter

Parameter Name	Parameter Type	Description
accelerometer	Vector3	Writes the accelerometer position vector (x, y, z) of the current controller into **accelerometer**.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    Vector3 accelerometer = new Vector3();
    int ret = controller. GetAccelerometer(ref accelerometer);
    if( ret == 0 ){
        Debug.Log(" Get controller accelerometer success! ");
    }else{
        Debug.Log(" Get controller accelerometer failed! ");
    }
}
```

4.7.11 IsTouchpadTouching

The **bool IsTouchpadTouching()** function is used to determine whether the current touchpad is touched.

Return Values

If the touchpad is being touched, **true** is returned. Otherwise, **false** is returned.

4.7.12 IsTouchpadTouchDown

The **bool IsTouchpadTouchDown()** function is used to determine whether a touch-down action is triggered on the touchpad. This function is compatible with the touch-down function of the touchpad in HuaweiVR1.0.

Return Values

If a touch-down action is triggered on the touchpad, **true** is returned. Otherwise, **false** is returned.

4.7.13 IsTouchpadTouchUp

The **bool IsTouchpadTouchUp()** function is used to determine whether a touch-up action is triggered on the touchpad. This function is compatible with the touch-up function of the touchpad in HuaweiVR1.0.

Return Values

If a touch-up action is triggered on the touchpad, **true** is returned. Otherwise, **false** is returned.

4.7.14 IsTouchpadSwipeUp

The **bool IsTouchpadSwipeUp()** function is used to determine whether a swipe-up action is triggered on the touchpad. This function is compatible with the swipe-up function of the touchpad in HuaweiVR1.0.

Return Values

If a swipe-up action is triggered on the touchpad, **true** is returned. Otherwise, **false** is returned.

4.7.15 IsTouchpadSwipeDown

The **bool IsTouchpadSwipeDown()** function is used to determine whether a swipe-down action is triggered on the touchpad. This function is compatible with the swipe-down function of the touchpad in HuaweiVR1.0.

Return Values

If a swipe-down action is triggered on the touchpad, **true** is returned. Otherwise, **false** is returned.

4.7.16 IsTouchpadSwipeLeft

The **bool IsTouchpadSwipeLeft()** function is used to determine whether a left-swipe action is triggered on the touchpad. This function is compatible with the left-swipe function of the touchpad in HuaweiVR1.0.

Return Values

If a left-swipe action is triggered on the touchpad, **true** is returned. Otherwise, **false** is returned.

4.7.17 IsTouchpadSwipeRight

The **bool IsTouchpadSwipeRight()** function is used to determine whether a right-swipe action is triggered on the touchpad. This function is compatible with the right-swipe function of the touchpad in HuaweiVR1.0.

Return Values

If a right-swipe action is triggered on the touchpad, **true** is returned. Otherwise, **false** is returned.

Examples

```
void Start(){
    if(null == controller){
```

```
            Debug.LogError(" Controller is null ");
        }
}
//Developers need to call functions in required positions.
void Update(){
        if (controller. IsTouchpadTouching ()) {
            Debug.Log (" Touchpad is touching ");
        }
        if (controller.IsTouchpadSwipeDown ()) {
            Debug.Log (" Touchpad is swipe down ");
        }else if(controller.IsTouchpadSwipeUp()){
            Debug.Log ("Touchpad is swipe up ");
        }else if(controller.IsTouchpadSwipeLeft()){
            Debug.Log ("Touchpad is swipe left ");
        }else if(controller.IsTouchpadSwipeRight()){
            Debug.Log ("Touchpad is swipe right ");
        }
        if(controller.IsTouchpadTouchUp()){
            Debug.Log (" Touchpad is touch Up ");
        }else if(controller.IsTouchpadTouchDown()){
            Debug.Log (" Touchpad is touch Down ");
        }
}
```

4.7.18 IsButtonPressed

The **bool IsButtonPressed(ButtonType button)** function is used to determine whether the controller button is pressed.

Parameter

Parameter Name	Parameter Type	Description
button	ButtonType For details, see section 2.6 "ButtonType."	Indicates the controller button type.

Return Values

If the related button is pressed, **true** is returned. Otherwise, **false** is returned.

4.7.19 IsButtonDown

The **bool IsButtonDown(ButtonType button)** function is used to determine whether a touch-down action is triggered on the controller button. When **ButtonType** is set to **ButtonBack**, this function is compatible with the touch-down function of the back button in HuaweiVR1.0.

Parameter

Parameter Name	Parameter Type	Description
button	ButtonType For details, see section 2.6 "ButtonType."	Indicates the controller button type.

NOTE

When ButtonType is **ButtonBack**, in HuaweiVR2.0, this interface cannot be invoked to respond to an event (such as an exit event) that needs to be executed by pressing the **ButtonBack** button, and the IsButtonUp (ButtonType button) interface response can be called to respond to such events. .

Return Values

If a touch-down action is triggered on the related button, **true** is returned. Otherwise, **false** is returned.

4.7.20 IsButtonUp

The **bool IsButtonUp(ButtonType button)** function is used to determine whether a touch-up action is triggered on the controller button. When **ButtonType** is set to **ButtonConfirm**, this function is compatible with the click function of the touchpad in HuaweiVR1.0. When **ButtonType** is set to **ButtonBack**, this function is compatible with the touch-up function of the back button in HuaweiVR1.0.

Parameter

Parameter Name	Parameter Type	Description
button	ButtonType For details, see section 2.6 "ButtonType."	Indicates the controller button type.

Return Values

If a touch-up action is triggered on the related button, **true** is returned. Otherwise, **false** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
```

```
}else{
    if(controller. IsButtonPressed (ButtonType.ButtonConfirm)){
        Debug.Log(" Confirm button is pressed! ");
    }
    if(controller.IsButtonDown (ButtonType.ButtonConfirm)){
        Debug.Log(" Confirm button is pressed down! ");
    }
    if(controller.IsButtonUp (ButtonType.ButtonConfirm)){
        Debug.Log(" Confirm button is pressed up! ");
    }
}
```

NOTE

- The **ButtonHome**, **ButtonVolumeInc**, and **ButtonVolumeDec** buttons on controllers are preconfigured. To avoid conflicts, **false** is returned when common applications obtain values of these buttons.
- When ButtonType is **ButtonBack**, in HuaweiVR2.0, this interface is called to respond to an event (such as an exit event) that needs to be executed by pressing the **ButtonBack** button.

4.7.21 StartVibration() (Not Supported)

The **int StartVibration()** function is used to enable the controller to start vibration.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    int ret = controller.StartVibration();
    if(ret == 0){
        Debug.Log(" Controller start vibration! ");
    }
}
```

4.7.22 StopVibration() (Not Supported)

int StopVibration() is used to enable the controller to stop vibration.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
if(null == controller){
    Debug.LogError(" Controller is null ");
}else{
    int ret = controller.StopVibration ();
    if(ret == 0){
            Debug.Log(" Controller stop vibration! ");
    }
}
```

4.8 IRenderHandle

The IRenderHandle class provides render-related functions.

Overview

This class is used to capture images provided by applications and enables dispersion to be displayed.

Public Methods

Method Name	Return Value Type	Description
CaptureEyeImage(String filePath)	int	Is used to capture images provided by applications.
EnableChromaticAberration(bool enable)	void	Enables or disables the dispersion function.
GetRenderStatics(ref RenderStatistics renderStatics)	int	Is used to obtain render statistic information.

4.8.1 CaptureEyeImage

The **int CaptureEyeImage(string filePath)** function is used to capture the images provided by applications and save the images in JPEG format into a file path, such as **/sdcard/captures/cp.jpeg**. Images can be captured continuously with this function called per frame at a time.

Parameter

Parameter Name	Parameter Type	Description
filePath	string	Indicates the image save path.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

NOTE

- Before this function is called in a non-script thread or application main thread, the current thread needs to be bound with the Java thread. If this function does not need to be called, the current thread needs to be unbound with the Java thread.
- Before using the screenshot capturing function, set the write permission on the Unity platform by choosing **Player Settings…** > **Other Settings** > **Write Permission** and selecting **External (SDCard)**.

Examples

```
int i = 0;
void Update(){
    i.ToString();
    string filePath = "/sdcard/captures/" + i + ".jpeg";
    int ret = renderHandle.CaptureEyeImage (filePath);
    i++;
    if(ret == 0){
          Debug.Log(" Capture Images Success! ");
    }else{
          Debug.Log(" Capture Images Failed! ");
    }
}
```

4.8.2 EnableChromaticAberration

The **void EnableChromaticAberration(bool enable)** function is used to enable or disable the dispersion function. If the value is **true**, the dispersion function is enabled. If the value is **false**, the dispersion function is disabled.

Parameter

Parameter Name	Parameter Type	Description
enable	bool	Enables the dispersion function. If the value is **true**, the dispersion function is enabled. If the value is **false**, the dispersion function is

		disabled.

NOTE

This function needs to be called in each frame of **Update()** to enable the dispersion function or be called in **Start()** or other functions by delaying four frames.

Examples

```
void Update(){
    renderHandle.EnableChromaticAberration(true);
}
```

4.8.3 GetRenderStatics

The **int GetRenderStatics(ref RenderStatistics renderStatics)** function is used to obtain rendering-related statistic information, such as the frame submit rate, frame rendering rate, and number of rendering timeout frames.

Parameter

Parameter Name	Parameter Type	Description
renderStatics	RenderStatistics For details, see section 3.3 "RenderStatistics."	Indicates the rendering statistic information.

Return Values

If the function is successfully called, **0** is returned. If the function fails to be called, **–1** is returned.

Examples

```
RenderStatistics renderStatics;
void Update(){
    renderHandle.GetRenderStatics (ref renderStatics);
    int submiteFrameRate = (int)renderStatics.SubmitFrameRate; //Frame submit rate
    int renderFrameRate = (int)renderStatics.RenderFrameRate; //Frame rendering rate
}
```

4.8.4 EnableMonocularDisplay

The void EnableMonocularDisplay(bool enable) function is used to enable the monocular rendering mode. If **enable** is true, monocular rendering mode is enabled, if **enable** is false, monocular rendering mode is disabled, and it is normal binocular rendering mode.

Parameter

Parameter Name	Parameter Type	Description
enable	bool	Enables the monocular function. If the value is **true**, monocular rendering mode is enabled. If the value is **false**, monocular rendering mode is disabled.

NOTE

- This function needs to be called before **Update()**.
- The monocular rendering mode has no parallax effect. Developers need to determine whether the mode needs to be enabled according to the scene.
- After the interface is called, subsequent scenes will maintain the current rendering mode.

Examples

```
void Start(){
    renderHandle.EnableMonocularDisplay (true);
}
```

4.9 HVRPluginCore

The HVRPluginCore class provides the function of setting the far/near tailored plane distance and the safe mode.

Overview

This class is used to set the far/near tailored plane distance, enable or disable the safe mode, and set the duration when the safe mode is enabled.

Public Static Methods

Method Name	Return Value Type	Description
HVR_SetClipPlaneParams(float m_nearClipPlane = 0.01f, float m_farClipPlane = 1000.0f)	void	Is used to set the far/near tailored plane distance.
HVR_OpenSafeMode()	void	Enables the safe mode.
HVR_CloseSafeMode()	void	Disables the safe

Method Name	Return Value Type	Description
		mode.
HVR_SetSafeModeParams(float durationSeconds)	void	Is used to set the duration when the safe mode is enabled.
HVR_SetSafeModeParams(byte mAlpha = 64, HVRDefCore.SafeModeColor mColor = HVRDefCore.SafeModeColor.Default, HVRDefCore.SafeModeColor mCentralColor = HVRDefCore.SafeModeColor.Default)	void	Is used to set the image transparency and color after the safe mode is enabled.

4.9.1 HVR_SetClipPlaneParams

The **void HVR_SetClipPlaneParams(float m_nearClipPlane, float m_farClipPlane)** function is used to dynamically set the far/near tailored plane distance.

Parameters

Parameter Name	Parameter Type	Description
m_nearClipPlane	float	Indicates the near tailored plane distance. The default value is 0.01.
m_farClipPlane	float	Indicates the far tailored plane distance. The default value is 1000.

Examples

```
float mNearClipPlane = 0.02f;
float mFarClipPlane=1200.0f;
HVRPluginCore.HVR_SetClipPlaneParams(mNearClipPlane, mFarClipPlane);
```

4.9.2 HVR_OpenSafeMode

The **void HVR_OpenSafeMode()** function is used to enable the safe mode. When a user Opens the smartphone camera and maps the data collected from the camera into the VR scene, the HuaweiVR1.0 helmet reminds the user of external environment.

Example

```
HVRPluginCore.HVR_OpenSafeMode();
```

NOTE

Only HuaweiVR1.0 supports this function.

4.9.3 HVR_CloseSafeMode

The **void HVR_CloseSafeMode ()** function is used to disable the safe mode. When a user closes the smartphone camera and cancels the mapping of the data collected from the camera into the VR scene, the HuaweiVR1.0 helmet cancels the reminding of the user of external environment.

Example

```
HVRPluginCore.HVR_CloseSafeMode;
```

NOTE

Only HuaweiVR1.0 supports this function.

4.9.4 HVR_SetSafeModeParams

The **void HVR_SetSafeModeParams(float durationSeconds)** function is used to set the duration for enabling the safe mode. The start time point is when the safe mode function is successfully called. After the set duration expires, the safe mode is automatically closed.

The default duration of enabling the safe mode is 10s. It is recommended that developers set this duration to less than 10s. If the duration is greater than 10s, the application running power consumption increases.

Parameter

Parameter Name	Parameter Type	Description
durationSeconds	float	Indicates the duration of enabling the safe mode. The unit is second.

Example

```
float durationSeconds = 10.0f; //The safe mode is automatically closed in after the safe mode is enabled about 10 seconds.
HVRPluginCore.HVR_SetSafeModeParams (durationSeconds);
```

NOTE

Only HuaweiVR1.0 supports this function.

The **HVR_SetSafeModeParams(byte mAlpha = 64, HVRDefCore.SafeModeColor mColor, HVRDefCore.SafeModeColor mCentralColor)** function is used to set the image transparency and color after the safe mode is enabled. The transparency can be changed to an appropriate value without affecting the scene visibility. In addition, the colors of the

intermediate area in the safe mode and the surrounding area can be changed at the same time. The intermediate area refers to the area to which users need to pay a special attention.

Parameters

Parameter Name	Parameter Type	Description
mAlpha	byte	Indicates the image transparency. **0** indicates complete transparency. The minimum value is 16. 255 indicates no transparency. The default value is 64.
mColor	HVRDefCore.SafeModeColor	Indicates the image color. For details, see section 4.3.1 "SafeModeColor."
mCentralColor	HVRDefCore.SafeModeColor	Indicates the color of an image in the intermediate area. For details, see section 4.3.1 "SafeModeColor."

Examples

```
byte mAlpha = 64;
HVRDefCore.SafeModeColor mColor = HVRDefCore.SafeModeColor.White;
HVRDefCore.SafeModeColor mCentralColor = HVRDefCore.SafeModeColor.Yellow;
HVRPluginCore. HVR_SetSafeModeParams(mAlpha, mColor, mCentralColor);
```

NOTE

Only HuaweiVR1.0 supports this function.

4.10 HvrEventListner

The HvrEventListner class provides static delegation variables to transfer to-be-listened UI input events of the gaze cursor. The events include the click, touch-down, touch-up, enter, and exit events.

Public Static Delegation Variables

Delegation Variable Name	Type	Description
onEnter	VoidDelegate	Indicates an enter event.
onExit	VoidDelegate	Indicates an exit event.
onDown	VoidDelegate	Indicates a touch-down event.
onUp	VoidDelegate	Indicates a touch-up event.
onClick	VoidDelegate	Indicates a click event.

The delegation variable type **VoidDelegate** is obtained from **public delegate void VoidDelegate (GameObject go)**.

NOTE

Only HuaweiVR1.0 supports this function.

Examples

```
//When the gaze cursor moves to the exit button, a user clicks the touchpad of the helmet and exits from the application. This script is bound with the exit button.
void Start () {
    HvrEventListner.Get(transform.gameObject).onClick = onPointClick;
}
private void onPointClick(GameObject go){
    if(go == transform.gameObject){
        Application.Quit ();
    }
}
```