6. NativeXR SDK API Interface Function List

6.1 Flow chart

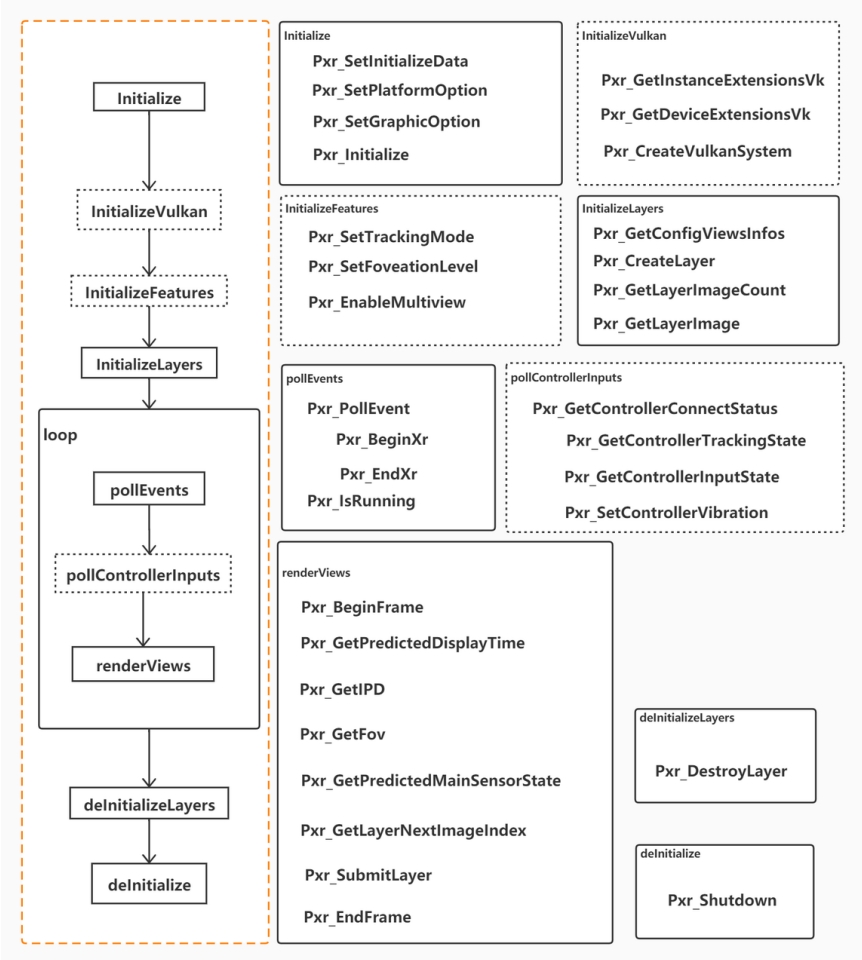

Figure6.1 Flow chart

The generalization of SDK process is as follows: init->render->shutdown, The initialization section includes setting up the platform and the graphics API, initializing layers, etc. Render includes beginFrame, get sens textureID, submit current frame, etc. The Shutdown section is mainly for resource release.

The SDK adopts an event mechanism. For example, Pxr_BeginXr can only be called after receiving XR_SESSION_STATE_READY; Pxr_EndXr can be called after receiving XR_SESSION_STATE_STOPPING, oth the call will fail.

6.2 Initialization related

- **Pxr_SetInitializeData**

Function name: void Pxr_SetInitializeData(PxrInitParamData* params)

Function: Set the data required for initialization, and it needs to be called before all other API.

Parameter:

```
typedef struct PxrInitParamData_ {
    void*   activity;         // activity handle, app->activity->clazz
```

```
    void*   vm;               //java vm handle, app->activity->vm
    int     controllerdof;    //1: controller 6dof, 0: controller 3dof
    int     headdof;          //1: headset 6dof, 0: headset 3dof
} PxrInitParamData;
```

Return value: None

Method of calling: Pxr_SetInitializeData(&initParamData)

- **Pxr_SetPlatformOption**

Function name: void Pxr_SetPlatformOption(PxrPlatformOption platform)

Function: Set the platform, and Native is set to PXR_NATIVE

Parameter: platform enum

Return value: None

Method of calling: Pxr_SetPlatformOption(PXR_NATIVE)

- **Pxr_SetGraphicOption**

Function name: void Pxr_SetGraphicOption(PxrGraphicOption graphic)

Function: Set the graphics API type, Vulkan or OpenGLES

Parameter: graphic: Graphics API type

Return value: None

Method of calling: Pxr_SetGraphicOption (*PXR_OPENGL_ES*)

- **Pxr_Initialize**

Function name: int Pxr_Initialize()

Function: SDK initialization

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_Initialize()

- **Pxr_IsInitialized**

Function name: bool Pxr_IsInitialized()

Function: Detects if the SDK is initialized.

Parameter: None

Return value: true-Initialized, false-Uninitialized

Method of calling: Pxr_IsInitialized()

6.3 Event related

- **Pxr_PollEvent**

Function name: bool Pxr_PollEvent(int eventCountMAX,int* eventDataCountOutput,

PxrEventDataBuffer** eventDataPtr)

Function: Get latest event

Parameter: eventCountMAX: Maximum number of messages

eventDataCountOutput: Number of messages received

eventDataPtr: Array of Event Structure

Return value: true-New event, false-No event

Method of calling: Pxr_PollEvent(MaxEventCount,eventCount,eventDataPointer)

6.4 Render related

- **Pxr_BeginXr**

Function name: int Pxr_BeginXr()

Function: Enter XR mode

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_BeginXr()

- **Pxr_IsRunning**

Function name: bool Pxr_IsRunning()

Function: Detects if XR mode has been entered.

Parameter: None

Return value: true-Entered, false-Not entered

Method of calling: Pxr_IsRunning ()

- **Pxr_BeginFrame**

Function name: int Pxr_BeginFrame()

Function: Start of each frame rendering.

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_BeginFrame()

- **Pxr_GetConfigViewsInfos**

Function name: int Pxr_GetConfigViewsInfos(uint32_t* maxImageRectWidth, uint32_t* maxImageRectHei
uint32_t* recommendedImageRectWidth, uint32_t* recommendedImageRectHeight);

Function: Get the maximum and recommended width and height of the RenderBuffer, in pixels.

Parameter: maxImageRectWidth: return maximum width

maxImageRectHeight: return maximum height

recommendedImageRectWidth: return recommended width

recommendedImageRectHeight: return recommended height

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetConfigViewsInfos(&maxImageRectWidth, &maxImageRectHeight,
&recommendedImageRectWidth, &recommendedImageRectHeight)

- **Pxr_CreateLayer**

Function name: int Pxr_CreateLayer(const PxrLayerParam* layerParam)

Function: Create a rendering layer

Parameter: Structure of the layer to be created

```
typedef struct PxrLayerParam_ {
    int               layerId;          // LayerID, 0,1,2,3,...
    PxrLayerShape     layerShape;       // Projection/Quad/Cylinder
    PxrLayerType      layerType;        // Overlay: Texture will be rendered over the
    PxrLayerLayout    layerLayout;      // Stereo: Stereo binocular layers, left and ri
    uint64_t          format;           // Image formats, for example, OpenGLES can be
```

```
    uint32_t          width;            // Image pixel width
    uint32_t          height;           // Image pixel height
    uint32_t          sampleCount;      // Number of image sampling channels, related
    uint32_t          faceCount;        // Number of image faces, generally 1 or 6
    uint32_t          arraySize;        // Number of image layers, generally 1, set to
    uint32_t          mipmapCount;      // Number of image mipmap
    uint32_t          layerFlags;       // Flag, refer to PxrLayerCreateFlags for more
    uint32_t          externalImageCount;    // used when the created layer uses extern
    uint64_t*         externalImages[2];      // used when the layer created uses an ex
} PxrLayerParam;
```

Return value: 0-Success, other-Failure

Method of calling: Pxr_CreateLayer(&layerParam)

- **Pxr_GetLayerImageCount**

Function name: int Pxr_GetLayerImageCount(int layerId, PxrEyeType eye, uint32_t* imageCount)

Function: Get the number of images corresponding to the layer

Parameter: layerId： layer id

eye： PXR_EYE_LEFT，PXR_EYE_RIGHT

imageCount：return the number of images

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetLayerImageCount(0, (PxrEyeType)i, &imageCounts)

- **Pxr_GetLayerImage**

Function name: int Pxr_GetLayerImage(int layerId, PxrEyeType eye, int imageIndex, uint64_t* image);

Function: Get the image handle of the corresponding serial number of the layer

Parameter:

layerId: layer id

eye: PXR_EYE_LEFT, PXR_EYE_RIGHT

imageIndex: the image handle number to be obtained

image: return the handle value of corresponding layer image. OpenGLES will return Texture ID; Vulkan will
VkImage.

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetLayerImage(0, (PxrEyeType)i, j, &LayerImages[i][j])

- **Pxr_GetLayerNextImageIndex**

Function name: int Pxr_GetLayerNextImageIndex(int layerId, int* imageIndex)

Function: Get the serial number of image handle to be rendered in the next frame of the layer

Parameter: layerId: layer id

imageIndex: return the serial number of image handle

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetLayerNextImageIndex(0, &imageIndex);

- **Pxr_GetLayerAndroidSurface**

Function name: int Pxr_GetLayerAndroidSurface(int layerId, PxrEyeType eye, jobject* androidSurface)

Function: Get the android surface created when the layer is created (When Pxr_CreateLayer, it is valid if the
layerFlags input contains PXR_LAYER_FLAG_ANDROID_SURFACE flags), it can be used directly for video n
decoding, etc.

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetLayerAndroidSurface(0, (PxrEyeType)i, &androidSuface)

- **Pxr_GetFov**

Function name: int Pxr_GetFov(PxrEyeType eye, float* fovLeft, float* fovRight, float* fovUp, float* fovDown

Function: Get fov

Parameter: eye： PXR_EYE_LEFT，PXR_EYE_RIGHT

fovLeft： Horizontal left fov

fovRight ： Horizontal right fov

fovUp： Vertical up fov

fovDown： Vertical down fov

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetFov(PxrEyeType(i),&fovL,&fovR,&fovU,&fovD);

- **Pxr_GetIPD**

Function name: float Pxr_GetIPD()

Function: Get ipd (interpupillary distance)

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: ipd = Pxr_GetIPD();

- **Pxr_SubmitLayer**

Function name: int Pxr_SubmitLayer(const PxrLayerHeader* layer)

Function: Commit the current rendering layer

Parameter: PxrLayerHeader Pointer. The actual object can be PxrLayerProjection/PxrLayerQuad/PxrLayerCylinder, etc.

```
typedef struct PxrLayerHeader_ {
    int          layerId;               // Layer ID
    uint32_t     layerFlags;            // Submit layer flag, refer to PxrLayerSubmitFla
    float        colorScale[4];         // Layer color scale and bias, when synthesized,
    float        colorBias[4];
    int          compositionDepth;      // The depth value of the layer composition, suc
    int          sensorFrameIndex;      // Index of sensor data used for layer rendering
    int          imageIndex;            // Image index used in layer rendering, this fie
    PxrPosef     headPose;              // If PXR_LAYER_FLAG_USE_EXTERNAL_HEAD_POSE flag
} PxrLayerHeader;
```

```
typedef struct PxrLayerProjection_ {  // For Render Buffer layer submission
    PxrLayerHeader    header;
    float             depth;              // The center point corresponds to the depth v
} PxrLayerProjection;
```

```
typedef struct PxrLayerQuad_ {   // For 2D planar layer submission
    PxrLayerHeader     header;
    PxrPosef           pose;          // Position and pose of 2D in the scene
    float              size[2];       // The width and height of the 2D in the scene, in
} PxrLayerQuad;
```

```
typedef struct PxrLayerCylinder_ {    // For column layer submission
    PxrLayerHeader     header;
```

```
    PxrPosef           pose;           // Position and pose of the cylinder in the scen
    float              radius;         // The radius of the horizontal cross-section of
    float              centralAngle;   // The radian of the horizontal cross-section of
    float              height;         // Height of cylinder, in meters
} PxrLayerCylinder;
```

Return value: 0-Success, other-Failure

Method of calling: Pxr_SubmitLayer((PxrLayerHeader*)&layerProjection)

- **Pxr_EndFrame**

Function name: int Pxr_EndFrame()

Function: End the current frame and submit all layers submitted by the rendering layer to the composition
Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_EndFrame()

- **Pxr_EndXr**

Function name: Pxr_EndXr()

Function: Exit XR mode

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_EndXr()

6.5 Sensor tracking related

- **Pxr_GetPredictedDisplayTime**

Function name: int Pxr_GetPredictedDisplayTime(double* predictedDisplayTimeMs)

Function: Get the predicted display time of the current render frame

Parameter: predictedDisplayTimeMs: Return the predicted display time of the current rendered frame, in milliseconds

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetPredictedDisplayTime(&predictedDisplayTimeMs);

- **Pxr_GetPredictedMainSensorState**

Function name: int Pxr_GetPredictedMainSensorState(double predictedDisplayTimeMs,

PxrSensorState* sensorState, int* sensorFrameIndex);

Function: Get the predicted position and pose of head

Parameter: predictTimeMs: Predicted display time of the current rendered frame, obtained by Pxr_GetPredictedDisplayTime

sensorState: Return the obtained position and pose of sensor and other data

sensorFrameIndex: Return the serial number corresponding to the pose and position of sensor

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetPredictedMainSensorState (predictedDisplayTimeMs, &sensorState, &sensorFrameIndex);

- **Pxr_GetPredictedMainSensorStateWithEyePose**

Function name: int Pxr_GetPredictedMainSensorStateWithEyePose(double predictedDisplayTimeMs,

PxrSensorState* sensorState, int* sensorFrameIndex,

int eyeCount, PxrPosef* eyePoses);

Function: Get the predicted position and pose data of head, and the corresponding eye pose

Parameter: predictTimeMs: The predicted display time of the current rendered frame, obtained by Pxr_GetPredictedDisplayTime

sensorState: return the sensor position, pose and other data obtained

sensorFrameIndex: return the serial number corresponding to the position and pose of sensor

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetPredictedMainSensorStateWithEyePose(predictedDisplayTimeMs, &sensorStat &sensorFrameIndex);

- **Pxr_ResetSensor**

Function name: int Pxr_ResetSensor(PxrResetSensorOption option)

Function: Reset pose

Parameter: reset options

PXR_RESET_POSITION: Reset position only

PXR_RESET_ORIENTATION: Reset pose only

PXR_RESET_ORIENTATION_Y_ONLY: reset only the pose of rotation around the Y axis

PXR_RESET_ALL: reset both position and pose

Return value: 0-Success, other-Failure

Method of calling: Pxr_ResetSensor(PXR_RESET_ALL)

6.6 Feature related

6.6.1 FFR

- **Pxr_SetFoveationLevel**

Function name: int Pxr_SetFoveationLevel(PxrFoveationLevel level)

Function: Set FFR level

Parameter: level：

Foveation_Level_NONE

Foveation_Level_LOW

Foveation_Level_MID

Foveation_Level_HIGH

Foveation_Level_TOP_HIGH

Foveation_Level_NONE： off

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetFoveationLevel(Foveation_Level_HIGH)

6.6.2 Eyetracking

- **Pxr_SetTrackingMode**

Function name: int Pxr_SetTrackingMode(PxrTrackingModeFlags trackingMode)

Function: Set the tracking mode

Parameter: trackingMode: tracking mode flag

PXR_TRACKING_MODE_ROTATION_BIT: enable pose tracking (open by default)

PXR_TRACKING_MODE_ROTATION_BIT: enable pose tracking (open by default)

PXR_TRACKING_MODE_POSITION_BIT: enable position tracking

PXR_TRACKING_MODE_EYE_BIT: enable eye tracking (devices that support eye tracking)

Return value: 0-Success, other-Failure Pxr_CreateLayer

Method of calling: Pxr_SetTrackingMode(PXR_TRACKING_MODE_ROTATION_BIT | PXR_TRACKING_MODE_POSITION_BIT | PXR_TRACKING_MODE_EYE_BIT)

- **Pxr_GetTrackingMode**

Function name: int Pxr_GetTrackingMode(PxrTrackingModeFlags* trackingMode)

Function: Get the value of tracking mode, a value of 7 means EyeTracking is supported

Parameter: trackingMode: the value of trackingMode obtained

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetTrackingMode(&mode)

- **Pxr_GetEyeTrackingData**

Function name: int Pxr_GetEyeTrackingData(PxrEyeTrackingData* eyeTrackingData)

Function: Get data for eye tracking

Parameter: eyeTrackingData

```
typedef struct PxrEyeTrackingData_ {
    int32_t      leftEyePoseStatus;            //Left eye pose data status
    int32_t      rightEyePoseStatus;           //Right eye pose data status
    int32_t      combinedEyePoseStatus;        //Combined eye pose data state
    float        leftEyeGazePoint[3];          //Origin coordinate of left eye gaze poin
    float        rightEyeGazePoint[3];         //Origin coordinate of right eye gaze poi
    float        combinedEyeGazePoint[3];      //Origin coordinate of combined eye gaze
    float        leftEyeGazeVector[3];         //Vector of left eye gaze point
    float        rightEyeGazeVector[3];        //Vector of right eye gaze point
    float        combinedEyeGazeVector[3];     //Vector of combined eye gaze point
    float        leftEyeOpenness;              //The openness/closeness of left eye take
    float        rightEyeOpenness;             //The openness/closeness of right eye
    float        leftEyePupilDilation;         //Pupil size of left eye, in millimeter
    float        rightEyePupilDilation;        //Pupil size of right eye, in millimeter
    float        leftEyePositionGuide[3];      //The left eye position guidance data in
    float        rightEyePositionGuide[3];     //The right eye position guidance data in
    float        foveatedGazeDirection[3];     //Gaze direction, in meters, based on the
    int32_t      foveatedGazeTrackingState;    //Status of the gaze direction
} PxrEyeTrackingData;
```

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetEyeTrackingData(&data)

6.6.3 MultiView

- **Pxr_EnableMultiview**

Function name: bool Pxr_EnableMultiview(bool enable)

Function: Whether to enable multiView

Parameter: enable Enable or disable

Return value: true-Success, false-Failure

Method of calling: Pxr_EnableMultiview(true)

6.6.4 Tracking Origin

- **Pxr_SetTrackingOrigin**

Function name: int Pxr_SetTrackingOrigin(PxrTrackingOrigin trackingOrigin)

Function: Set tracking origin mode

Parameter: trackingOrigin:

PXR_EYE_LEVEL

PXR_FLOOR_LEVEL

PXR_STAGE_LEVEL

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetTrackingOrigin (PXR_FLOOR_LEVEL)

- **Pxr_GetTrackingOrigin**

Function name: int Pxr_GetTrackingOrigin(PxrTrackingOrigin* trackingOrigin)

Function: Get tracking origin settings

Parameter: trackingOrigin

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetTrackingOrigin (&origin)

6.6.5 ColorSpace

- **Pxr_SetColorSpace**

Function name: int Pxr_SetColorSpace(PxrColorSpace colorSpace)

Function: Set color space

Parameter: colorSpace

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetColorSpace (ColorSpaceSRGB)

6.7 Controller related

- **Pxr_GetControllerCapabilities**

Function name: int Pxr_GetControllerCapabilities(uint32_t deviceID, PxrControllerCapability *capability)

Function: Get controller capabilities

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT capability Controller capabiliti return, including controller type, 3DOF/6DOF, binding status and PxrControllerAbilities

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetControllerCapabilities(hand, &cap);

- **Pxr_GetControllerConnectStatus**

Function name: int Pxr_GetControllerConnectStatus(uint32_t deviceID)

Function: Get the connection status of controller

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT

Return value: 0-connected, 1-not connected

Method of calling: Pxr_GetControllerConnectStatus(hand)

- **Pxr_GetControllerInputEvent**

Function name: int Pxr_GetControllerInputEvent(uint32_t deviceID, PxrControllerInputEvent *event)

Function: Get event of controller buttons

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT

event Controller button event structure

Return value: 0-Success, other-Failure

Method of calling: **Pxr_GetControllerInputEvent** (hand,&event)

- **Pxr_ GetControllerInputState**

Function name: int Pxr_GetControllerInputState(uint32_t deviceID, PxrControllerInputState *state)

Function: Get state of controller button

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT

state Controller button status structure

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetControllerInputState(hand,&state)

- **Pxr_GetControllerTrackingState**

Function name: int Pxr_GetControllerTrackingState(uint32_t deviceID, double predictTime,

float headSensorData[], PxrControllerTracking *tracking);

Function: Get posture of controller

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT

headSensorData Input headset posture data

tracking Output controller posture data

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetControllerTrackingState(hand,PredictedDisplayTime,sensor,&tracking)

- **Pxr_SetControllerMainInputHandle**

Function name: int Pxr_SetControllerMainInputHandle(uint32_t deviceID)

Function: Set the main controller

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetControllerMainInputHandle（PXR_CONTROLLER_LEFT）

- **Pxr_SetControllerEnableKey**

Function name: int Pxr_SetControllerEnableKey(bool isEnable,PxrControllerKeyMap Key)

Function: Whether to disable key

Parameter: isEnable true-enable key, false-disable key

Key Corresponding keys

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetControllerEnableKey (false, PXR_CONTROLLER_KEY_AX)

- **Pxr_SetControllerVibration**

Function name: int Pxr_SetControllerVibration(uint32_t deviceID, float strength, int time)

Function: Set controller vibration

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT strength Vibration Strength, ra
0.0f-1.0f

time Vibration time, range 0--65535 ms

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetControllerVibration(hand, trigger, 20)

- **Pxr_GetControllerinfo**

Function name: int Pxr_GetControllerinfo(uint32_t deviceID, PxrControllerInfo *info)

Function: Get controller information data

Parameter: deviceID PXR_CONTROLLER_LEFT，PXR_CONTROLLER_RIGHT info Controller information da
Bluetooth address, controller type version number, etc.

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetControllerinfo (&info)

6.8 Safety Boundary related

- **Pxr_GetBoundaryConfigured**

Function name: bool Pxr_GetBoundaryConfigured()

Function: Return result of Safety Boundary configuring

Parameter: None

Return value: true-Success, false-Failure

Method of calling: Pxr_GetBoundaryConfigured()

- **Pxr_GetBoundaryEnabled**

Function name: bool Pxr_GetBoundaryEnabled()

Function: Get the enabling status of safety zone protection system

Parameter: None

Return value: true-Success, false-Failure

Method of calling: Pxr_GetBoundaryEnabled()

- **Pxr_SetBoundaryVisible**

Function name: int Pxr_SetBoundaryVisible(bool value)

Function: Force to set whether Safety Boundary is visible (Note: Safety Boundary activation and user configuration in system settings will overwrite this interface's action)

Parameter: value: whether the Safety Boundary is visible or not

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetBoundaryVisible()

- **Pxr_GetBoundaryVisible**

Function name: bool Pxr_GetBoundaryVisible()

Function: Get whether the safety boundary is visible or not

Parameter: None

Return value: true-Visible；false-Invisible Method of calling: Pxr_GetBoundaryVisible()

- **Pxr_TestNodeIsInBoundary**

Function name: int Pxr_TestNodeIsInBoundary(PxrBoundaryTestNode node, bool isPlayArea,

PxrBoundaryTriggerInfo* info);

Function: Return testing results of tracking nodes to specific boundary types

Parameter: node: tracking node PXR_BOUNDARY_TEST_NODE_LEFT_HAND，

PXR_BOUNDARY_TEST_NODE_RIGHT_HAND，

PXR_BOUNDARY_TEST_NODE_HEAD

isPlayArea：whether it is an inscribed rectangular area (when the value is false, it means an external cust
boundary) info: : return the test result structure

```
typedef struct PxrBoundaryTriggerInfo_ {
    bool            isTriggering;           // If boundary is triggered
    float           closestDistance;        // Minimum distance of tracking nodes and
    PxrVector3f     closestPoint;           // Closest point of tracking nodes and bo
    PxrVector3f     closestPointNormal;     // Normal of closest point
    bool            valid;                  // If the result is valid
} PxrBoundaryTriggerInfo;
```

Return value: 0-Success, other-Failure

Method of calling: Pxr_TestNodeIsInBoundary(PXR_BOUNDARY_TEST_NODE_HEAD, true, &info)

- **Pxr_TestPointIsInBoundary**

Function name: Pxr_TestPointIsInBoundary(const PxrVector3f* point, bool isPlayArea,

PxrBoundaryTriggerInfo* info)

Function: Return testing results of a 3-dimensional point coordinate to a specific boundary type

Parameter: point：coordinate of point

isPlayArea: whether it is an inscribed rectangular area (when the value is false, it means an external custo
boundary)

info: : return the test result structure

typedef struct PxrBoundaryTriggerInfo_ {

bool isTriggering; // If boundary is triggered

float closestDistance; // Minimum distance of tracking nodes and boundary

PxrVector3f closestPoint; // Closest point of tracking nodes and boundary

PxrVector3f closestPointNormal;// Normal of closest point

bool valid; // If the result is valid

}PxrBoundaryTriggerInfo

Return value: 0-Success, other-Failure

Method of calling: Pxr_TestNodeIsInBoundary(point, true, &info)

- **Pxr_GetBoundaryGeometry**

Function name: int Pxr_GetBoundaryGeometry(bool isPlayArea, uint32_t pointsCountInput, uint32_t*
pointsCountOutput, PxrVector3f* outPoints)

Function: Return the collection of boundary points

Parameter: isPlayArea: whether it is an inscribed rectangular area (when the value is false, it means an ext
custom boundary)

pointsCountInput: The number of points expected to be acquired, when its value is 0, indicates that indica
function is called to obtain the number of points that can be obtained.

pointsCountOutput: The number of points actually obtained outpoints： Points obtained

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetBoundaryGeometry(true, 0, &pointsCountOutput, NULL)

Pxr_GetBoundaryGeometry(true, pointsCountInput, &pointsCountOutput, outPoints)

- **Pxr_GetBoundaryGeometry2**

Function name: int Pxr_GetBoundaryGeometry2(bool isPlayArea, float ** outPointsFloat,

uint32_t *pointsCountOutput)

Function: Return the collection of boundary points

Parameter: isPlayArea: whether it is an inscribed rectangular area (when the value is false, it means an ext
custom boundary) outPointsFloat： Points obtained, float (*outPointsFloat)[3]

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetBoundaryGeometry2 (true, outPointsFloat, & pointsCountOutput)

- **Pxr_GetDialogState**

Function name: int Pxr_GetDialogState()

Function: Get the boundary dialog state

Parameter: None

Return value:

NothingDialog = -1,

GobackDialog = 0,

ToofarDialog = 1,

LostDialog = 2,

LostNoReason = 3,

LostCamera = 4,

LostHighLight = 5,

LostLowLight = 6,

LostLowFeatureCount = 7,

LostReLocation = 8

Method of calling: Pxr_GetDialogState()

6.9 SeeThrough Camera related

- **Pxr_GetSeeThroughData**

Function name: int Pxr_GetSeeThroughData(PxrSeeThoughData* data)

Function: Get SeeThrough Camera image

Parameter: data： SeeThrough Camera image structure

```
typedef struct PxrSeeThoughData_ {
    uint64_t        leftEyeTextureId;     //The left eye holds the image handle of seeth
                                          //Inputting 0 means that left eye image is not
    uint64_t        rightEyeTextureId;    //The right eye holds the image handle of seeth
                                          //Inputting 0 means that right eye image is no
    uint32_t        width;                //Inputting is the expected image width, outpu
                                          //Width, in pixels
    uint32_t        height;               //Inputting is the expected image height, outp
                                          //Height, in pixels
    uint32_t        exposure;             //Exposure time obtained
    int64_t         startTimeOfExposure;  //start time of exposure obtained
    bool            valid;                //Whether the obtained data is valid or not
}PxrSeeThoughData;
```

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetSeeThroughData(&data)

6.10 Performance related

- **Pxr_GetDisplayRefreshRatesAvailable**

Function name: int Pxr_GetDisplayRefreshRatesAvailable(uint32_t* count, float** rateArray)

Function: Get the available screen refresh rate

Parameter: count: the number of screen refresh rates that can be used

rateArray: available screen refresh rate

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetDisplayRefreshRatesAvailable(&count, &rateArray) // float* rateArray

- **Pxr_SetDisplayRefreshRate**

Function name: int Pxr_SetDisplayRefreshRate(float refreshRate)

Function: Set the screen refresh rate

Parameter: refreshRate: the screen refresh rate needs to be set

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetDisplayRefreshRate (72.0f)

- **Pxr_GetDisplayRefreshRate**

Function name: int Pxr_GetDisplayRefreshRate(float* refreshRate)

Function: Get the current screen refresh rate

Parameter: refreshRate: output screen refresh rate

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetDisplayRefreshRate (&rate)

6.11 Vulkan related

- **Pxr_GetDeviceExtensionsVk**

Function name: int Pxr_GetDeviceExtensionsVk(const char** extensionNamesArray,

uint32_t* extensionCount)

Function: Get the vulkan extension name supported by the device

Parameter: extensionNamesArray: Array of vulkan extension names supported by the obtained device

extensionCount： The number of vulkan extensions supported by the obtained device

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetDeviceExtensionsVk(NULL, &deviceExtensionsCount);

Pxr_GetDeviceExtensionsVk(deviceExtensions.data(), &deviceExtensionsCount)

- **Pxr_GetInstanceExtensionsVk**

Function name: int Pxr_GetInstanceExtensionsVk(const char** extensionNamesArray,

uint32_t* extensionCount) Function: Get the vulkan extension name supported by the instance

Parameter: extensionNamesArray: Array of vulkan extension names supported by the obtained instance

extensionCount: The number of vulkan extensions supported by the obtained instance

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetInstanceExtensionsVk(NULL, &instanceExtensionCount);

Pxr_GetInstanceExtensionsVk(extensions.data(), &instanceExtensionCount)

- **Pxr_CreateVulkanSystem**

Function name: int Pxr_CreateVulkanSystem(const PxrVulkanBinding* vulkanBinding)

Function: Inputting vulkan binding parameters to create a vulkan session

Parameter: vulkanBinding：PxrVulkanBinding vulkan binding structure

Return value: 0-Success, other-Failure

Method of calling: Pxr_CreateVulkanSystem(&vulkanBinding);

6.12 Configuration item related

```
typedef enum
{
      PXR_RENDER_TEXTURE_WIDTH = 0,      //eyebuffer width,  int, readonly
      PXR_RENDER_TEXTURE_HEIGHT,         //eyebuffer height, int, readonly
      PXR_SHOW_FPS,                      //fps whether to show, int, readonly
      PXR_RUNTIME_LOG_LEVEL,             //runtime loglevel, int,  readonly
      PXR_PXRPLUGIN_LOG_LEVEL,           //plugin loglevel,  int,  readonly
      PXR_UNITY_LOG_LEVEL,               //unity  loglevel,  int,  readonly
      PXR_UNREAL_LOG_LEVEL,              //unreal loglevel,  int,  readonly
      PXR_NATIVE_LOG_LEVEL,              //native loglevel,  int,  readonly
      PXR_TARGET_FRAME_RATE,             // target fps, int, readonly
      PXR_NECK_MODEL_X,                  //neck model x-coordinate, float, readonly
      PXR_NECK_MODEL_Y,                  //neck model y-coordinate, float, readonly
      PXR_NECK_MODEL_Z,                  //neck model z-coordinate, float, readonly
      PXR_DISPLAY_REFRESH_RATE,          //System screen refresh rate, float, readonly
      PXR_ENABLE_6DOF,                   //enable switch of 6dof, int, readwrite
      PXR_CONTROLLER_TYPE,               //current controller type, int, readonly
      PXR_PHYSICAL_IPD,                  //physical interpupilary distance, float, reador
      PXR_TO_DELTA_SENSOR_Y,             //delta y data, float, readonly
      PXR_GET_DISPLAY_RATE,              //screen refresh rate already set, float, readw
      PXR_FOVEATION_SUBSAMPLED_ENABLED,  //ffr subsample enable, int, readonly
      PXR_TRACKING_ORIGIN_HEIGHT,        //tracking origin height, float, readonly
      PXR_ENGINE_VERSION,                //current engine version, string, writeonly
      PXR_UNREAL_OPENGL_NOERROR,         //mark of openglnoerror, int, writeonly
      PXR_ENABLE_CPT,                    //content protection mark of projecting, int, w
      PXR_MRC_TEXTURE_ID,                //mrc uses texture handles, uint64, writeonly
      PXR_RENDER_FPS,                    //current rendering frame rate, float, readonly
      PXR_MSAA_LEVEL_RECOMMENDED         //recommended value of msaa level, int, readonl
}PxrConfigType;
```

- **Pxr_GetConfigInt**

Function name: int Pxr_GetConfigInt(PxrConfigType configIndex, int* configData)

Function: Get the value of the relevant configuration item

Parameter: configIndex: configuration item

configData: return the value of the configuration item

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetConfigInt (PXR_RENDER_TEXTURE_WIDTH， &data)

- **Pxr_GetConfigFloat**

Function name: int Pxr_GetConfigFloat(PxrConfigType configIndex, float* configData)

Function: Get the value of the relevant configuration item

Parameter: configIndex: configuration item

configData: return the value of the configuration item

Return value: 0-Success, other-Failure

Method of calling: Pxr_GetConfigFloat(PXR_RENDER_FPS, &data)

- **Pxr_SetConfigInt**

- **Pxr_SetConfigInt**

Function name: int Pxr_SetConfigInt(PxrConfigType configIndex, int configData)

Function: Set the value of the relevant configuration item

Parameter: configIndex: configuration item

configData:the value of the configuration item

Return value: 0-Success, other-Failure

Method of calling: Pxr_SetConfigInt (PXR_ENABLE_6DOF， 1)

6.13 End

- **Pxr_DestroyLayer**

Function name: int Pxr_DestroyLayer(int layerId)

Function: Destroy the built layer

Parameter: layerId: layer id

Return value: 0-Success, other-Failure

Method of calling: Pxr_DestroyLayer(eyeLayerId);

- **Pxr_Shutdown**

Function name: int Pxr_Shutdown()

Function: Exit and release resources

Parameter: None

Return value: 0-Success, other-Failure

Method of calling: Pxr_Shutdown()