|  |  |
| --- | --- |
| Product name | Confidentiality level |
|  |  |
| Product version | Total 33 pages 共 33页 |
|  |

HUAWEI AR Engine NDK Interface Manual

|  |  |  |  |
| --- | --- | --- | --- |
| Prepared by  拟制 | Jiangdongsheng/00428315  Qianpeng/00227536 | Date  日期 | 2018-05-01 |
| Reviewed by  审核 | Guozejin/00168222 | Date  日期 | 2018-10-31 |
| Approved by  批准 |  | Date  日期 |  |



Huawei Technologies Co., Ltd.

All rights reserved

Revision Record

| Date | Revision Version | CR ID / Defect ID CR | Sec No. | Change Description | Author |
| --- | --- | --- | --- | --- | --- |
| 2018/04/18 | 0.10 |  |  | Initialize this manual | Jiangdongsheng /00428315 |
| 2018/05/10 | 0.20 |  |  | Add check and set engnines module | Jiangdongsheng /00428315 |
| 2018/05/15 | 0.30 |  |  | Add HwArEnginesSelector\_getCreatedEngine function to acqure the current used engine type | Jiangdongsheng /00428315 |
| 2018/05/28 | 0.40 |  |  | Add some coonfig functions for HUAWEI AR Engine | Jiangdongsheng /00428315 |
| 2018/06/05 | 0.50 |  |  | Add BodyAR API | Jiangdongsheng /00428315 |
| 2018/06/18 | 0.60 |  |  | Add some instructions. | Jiangdongsheng /00428315 |
| 2018/06/28 | 0.70 |  |  | Add handAR API | Jiangdongsheng /00428315 |

Catalog 目 录

1 简介 5

1.1 Intetion 5

1.2 Scope 5

2 HUAWEI AR Engine NDK Interface 6

2.1 Process 6

2.2 Interface Introduction 6

2.2.1 Concepts 6

2.2.1.1. Dual Engine Support 6

2.2.1.2. Object ownership 7

2.2.1.3. Poses and Coordinate Spaces 7

2.2.1.4. World Coordinate Space 7

2.2.2 Modules 8

2.2.2.1. Common Definitions 8

2.2.2.2. HwArAnchor 9

2.2.2.3. HwArLightEstimate 10

2.2.2.4. HwArPlane 11

2.2.2.5. HwArPoint 13

2.2.2.6. HwArPointCloud 13

2.2.2.7. HwArPose 14

2.2.2.8. HwArTrackable 15

2.2.2.9. HwArEnginesSelector 16

2.2.2.10. HwArHand 17

2.2.2.11. HwArBody 19

2.2.2.12. HwArEnginesApk 21

2.2.2.13. HwArConfiguration 22

2.2.2.14. HwArSession 26

2.2.2.15. HwArCamera 28

2.2.2.16. HwArCameraConfig 28

2.2.2.17. HwArFrame 29

2.2.2.18. HwArHitResult 30

Figure List

图1 HUAWEI AR Engine NDK简要使用过程 6

# 简介

## Intetion

This manual describe the interface of Huawei AR NDK, and it is applicable to developers who will adopt the HUAWEI AR Engine service in their applications.

## Scope

This manual includes the process of HUAWEI AR Engine NDK, functionalities, parameters and returns.

# HUAWEI AR Engine NDK Interface

## Process

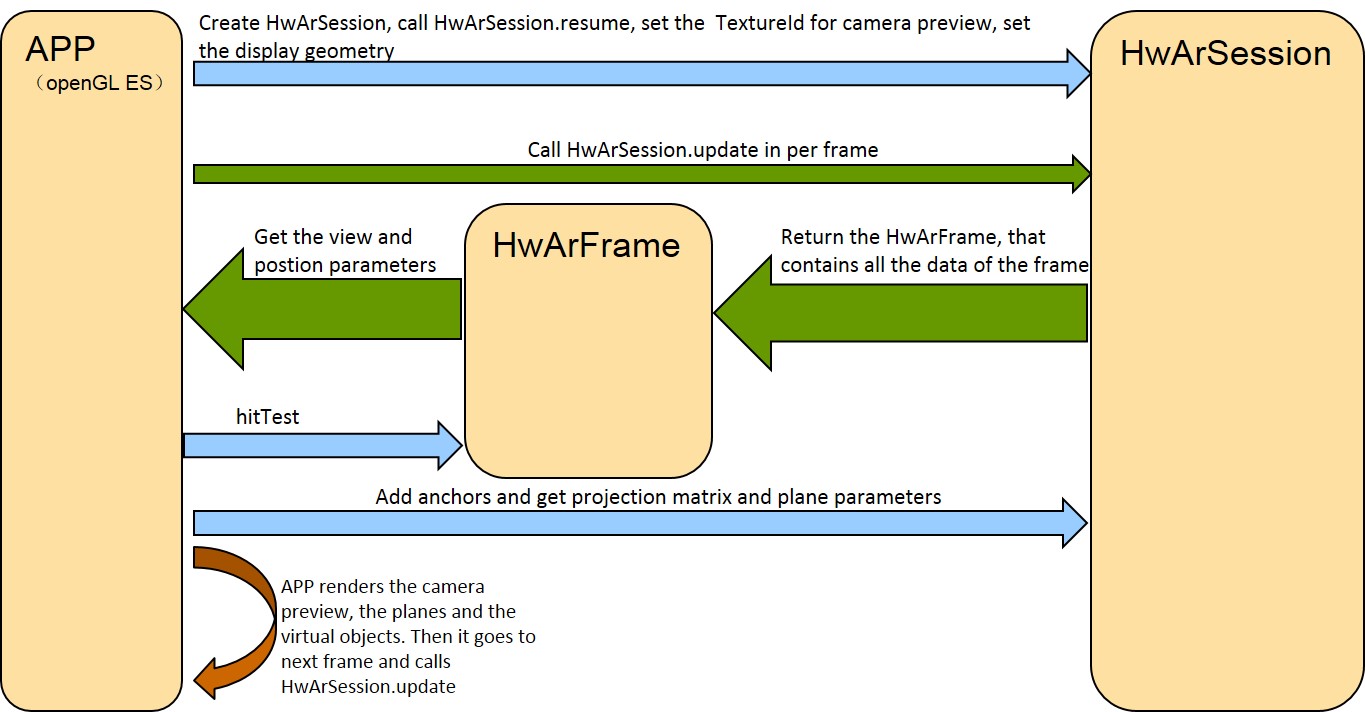


Fig 1. Brief introduction of HUAWEI AR Engine process

## Interface Introduction

### Concepts

#### Dual Engine Support

##### The HUAWEI AR Engine provides support for Google ARCore and Huawei's self-developed HUAWEI AR Engine through the same NDK API. The user can obtain the type of the engine supported by the terminal by using the HwArEnginesSelector\_checkAllAvailableEngines() method before using the AR ability. The method also automatically initializes a native supported engine according to the engine type supported by the terminal (if it is the dual engine supports, Huawei own engine is initialized). On the terminal device supported by the dual engine at the same time, the engine type can be specified by explicitly calling the HwArEnginesSelector\_setAREngine() method; if the terminal does not have a supported engine type, an exception will be thrown to end the program; if only one of the engines is supported, this engine will be set by default. At this time, the HwArEnginesSelector\_setAREngine() method cannot specify engines successfully, and only the engine supported by the device can be set. The user can also get the currently used engine type through HwArEnginesSelector\_getCreatedEngine().

#### Object ownership

##### HUAWEI AR Engine has two categories of objects: "value types" and "reference types".

##### Value types are owned by application. They are created and destroyed using the create / destroy methods, and are populated by HUAWEI AR Engine using methods with get in the method name.

##### Reference types are owned by HUAWEI AR Engine. A reference is acquired by one of the acquire methods. For each call to the acquire method, the application must call the matching release method. Note that even if last reference is released, HUAWEI AR Engine may continue to hold a reference to the object at HUAWEI AR Engine 's discretion.

Reference types are further split into:

##### Long-lived objects. These objects persist across frames, possibly for the life span of the application or session. Acquire may fail if HUAWEI AR Engine is in an incorrect state, e.g. not tracking. Acquire from list always succeeds, as the object already exists.

##### Transient large data. These objects are usually acquired per-frame and are a limited resource. The acquire call may fail if the resource is exhausted (too many are currently held), deadline exceeded (the target of the acquire was already released), or the resource is not yet available.

##### Note: Lists are value types (owned by application), but can hold references to long-lived objects. This means that the references held by a list are not released until either the list is destroyed, or is re-populated by another api call.

##### For example, HwArAnchorList, which is a value type, will hold references to HwArAnchors, which are long-lived objects.

#### Poses and Coordinate Spaces

##### A HwArPose describes a rigid transformation from one coordinate space to another. As provided from all HUAWEI AR Engine APIs, Poses always describe the transformation from object's local coordinate space to the world coordinate space (see below). That is, Poses from HUAWEI AR Engine APIs can be thought of as equivalent to OpenGL model matrices.

##### The transformation is defined using a quaternion rotation about the origin followed by a translation.

##### The coordinate system is right-handed, like OpenGL conventions.

##### Translation units are meters.

#### World Coordinate Space

As HUAWEI AR Engine's understanding of the environment changes, it adjusts its model of the world to keep things consistent. When this happens, the numerical location (coordinates) of the camera and anchors can change significantly to maintain appropriate relative positions of the physical locations they represent.

These changes mean that every frame should be considered to be in a completely unique world coordinate space. The numerical coordinates of anchors and the camera should never be used outside the rendering frame during which they were retrieved. If a position needs to be considered beyond the scope of a single rendering frame, either an anchor should be created or a position relative to a nearby existing anchor should be used.

### Modules

#### Common Definitions

###### HwArTrackingState

###### Illustration: Describes the tracking state of a Trackable.

|  |  |
| --- | --- |
| **HwArTrackingState** | **Description** |
| HWAR\_TRACKING\_STATE\_TRACKING | This state means the object is being tracked and it's state is valid. |
| HWAR\_TRACKING\_STATE\_PAUSED | This state indicates that HUAWEI AR Engine has paused tracking, and the related data is not accurate. |
| HWAR\_TRACKING\_STATE\_STOPPED | This state means that HUAWEI AR Engine has stopped tracking, and will never resume tracking. |

###### HwArStatus

###### Illustration:Return code indicating success or failure of a method.

|  |  |
| --- | --- |
| **HwArStatus** | **Description** |
| HWAR\_SUCCESS | The operation was successful. |
| HWAR\_ERROR\_INVALID\_ARGUMENT | One of the arguments was invalid, either null or not appropriate for the operation requested. |
| HWAR\_ERROR\_FATAL | An internal error occurred that the application should not attempt to recover from |
| HWAR\_ERROR\_SESSION\_PAUSED | An operation was attempted that requires the session be running, but the session was paused. |
| HWAR\_ERROR\_SESSION\_NOT\_PAUSED | An operation was attempted that requires the session be paused, but the session was running. |
| HWAR\_ERROR\_NOT\_TRACKING | An operation was attempted that the session be in the TRACKING state, but the session was not. |
| HWAR\_ERROR\_TEXTURE\_NOT\_SET | A texture name was not set by calling HwArSession\_setCameraTextureName() before the first call to HwArSession\_update() |
| HWAR\_ERROR\_MISSING\_GL\_CONTEXT | An operation required GL context but one was not available. |
| HWAR\_ERROR\_UNSUPPORTED\_CONFIGURATION | The configuration supplied to HwArSession\_configure() was unsupported. |
| HWAR\_ERROR\_CAMERA\_PERMISSION\_NOT\_GRANTED | The android camera permission has not been granted prior to calling HwArSession\_resume(). |
| HWAR\_ERROR\_DEADLINE\_EXCEEDED | Acquire failed because the object being acquired is already released. |
| HWAR\_ERROR\_RESOURCE\_EXHAUSTED | There are no available resources to complete the operation. |
| HWAR\_ERROR\_NOT\_YET\_AVAILABLE | Acquire failed because the data isn't available yet for the current frame. |
| HWAR\_ERROR\_CAMERA\_NOT\_AVAILABLE | The android camera has been reallocated to a higher priority app or is otherwise unavailable. |
| HWAR\_UNAVAILABLE\_ARSERSERVICE\_NOT\_INSTALLED | The engine service APK is not installed on this device. |
| HWAR\_UNAVAILABLE\_DEVICE\_NOT\_COMPATIBLE | The device is not currently compatible with the engine. |
| HWAR\_UNAVAILABLE\_APK\_TOO\_OLD | The service APK currently installed on device is too old and needs to be updated. |
| HWAR\_UNAVAILABLE\_SDK\_TOO\_OLD | The service APK currently installed no longer supports the HUAWEI AR Engine SDK that the application was built with. |
| HWAR\_UNAVAILABLE\_USER\_DECLINED\_INSTALLATION | The user declined installation of the service APK during this run of the application and the current request was not marked as user-initiated. |
| HWAR\_UNAVAILABLE\_EMUI\_NOT\_COMPATIBLE | The EMUI version is not currently compatible with the engine. |

#### HwArAnchor

##### Description: Describes a fixed location and orientation in the real world. To stay at a fixed location in physical space, the numerical description of this position will update as HUAWEI AR Engine's understanding of the space improves.

##### Typedefs:

###### HwArAnchor

###### Illustration: A position in space attached to a trackable.

###### [ArAnchor](https://developers.google.com/ar/reference/c/group/anchor#group__anchor_1gabb7df250c42a452f8360118ceb0f5e0a)List

###### Illustration: A list of anchors.

##### Functions:

###### void HwArAnchorList\_acquireItem(const HwArSession \*session,const HwArAnchorList \*anchor\_lis,int32\_t index,HwArAnchor \*\*out\_anchor)

###### Illustration: Acquires a reference to an indexed entry in the list.

###### void HwArAnchorList\_create(const HwArSession \*session,HwArAnchorList \*\*out\_anchor\_list)

###### Illustration: Creates an anchor list object.

###### void HwArAnchorList\_destroy(HwArAnchorList \*anchor\_list)

###### Illustration: Releases the memory used by an anchor list object, along with all the anchor references it holds.

###### void HwArAnchorList\_getSize(const HwArSession \*session,const HwArAnchorList \*anchor\_list,int32\_t \*out\_size)

###### Illustration: Retrieves the number of anchors in this list.

###### void HwArAnchor\_detach(HwArSession \*session,HwArAnchor \*anchor)

###### Illustration: Tells HUAWEI AR Engine to stop tracking and forget this anchor.

###### void HwArAnchor\_getPose(const HwArSession \*session,const HwArAnchor \*anchor,HwArPose \*out\_pose)

###### Illustration: Retrieves the pose of the anchor in the world coordinate space. This pose produced by this call may change each time Hw[ArSession\_update()](https://developers.google.com/ar/reference/c/group/session.html#group__session_1gad969eda706aff9e201bad0965ddc4589) is called. This pose should only be used for rendering if HwArAnchor\_getTrackingState() returns HWAR\_TRACKING\_STATE\_TRACKING.

###### void HwArAnchor\_getTrackingState(const HwArSession \*session,const HwArAnchor \*anchor,HwArTrackingState \*out\_tracking\_state)

###### Illustration: Retrieves the current state of the pose of this anchor.

###### void HwArAnchor\_release(HwArAnchor \*anchor)

###### Illustration: Releases a reference to an anchor. This does not mean that the anchor will stop tracking, as it will be obtainable from e.g. HwArSession\_getAllAnchors() if any other references exist.

#### HwArLightEstimate

##### Description: Holds information about the estimated lighting of the real scene.

##### Typedefs:

###### ArLightEstimateState

###### Illustration: Tracks the validity of a light estimate.

|  |  |
| --- | --- |
| **ArLightEstimateState** | **Desciption** |
| AR\_LIGHT\_ESTIMATE\_STATE\_NOT\_VALID | The light estimate is not valid this frame and should not be used for rendering. |
| AR\_LIGHT\_ESTIMATE\_STATE\_VALID | The light estimate is valid this frame. |

###### ArLightEstimate

###### Illustration: An estimate of the real-world lighting (value type).

##### Functions:

###### void HwArLightEstimate\_create(const HwArSession \*session,HwArLightEstimate \*\*out\_light\_estimate)

###### Illustration: Allocates a light estimate object.

###### void HwArLightEstimate\_destroy(HwArLightEstimate \*light\_estimate)

###### Illustration: Releases the provided light estimate object.

###### void HwArLightEstimate\_getPixelIntensity(const HwArSession \*session,const HwArLightEstimate \*light\_estimate,float \*out\_pixel\_intensity)

###### Illustration: Retrieves the pixel intensity, in gamma space, of the current camera view. Values are in the range (0.0, 1.0), with zero being black and one being white.

###### void HwArLightEstimate\_getState(const HwArSession \*session,const HwArLightEstimate \*light\_estimate,HwArLightEstimateState \*out\_light\_estimate\_state)

###### Illustration: Retrieves the validity state of a light estimate. If the resulting value of \*out\_light\_estimate\_state is not HWAR\_LIGHT\_ESTIMATE\_STATE\_VALID, the estimate should not be used for rendering.

#### HwArPlane

##### Description: Describes the current best knowledge of a real-world planar surface.

##### Typedefs:

###### HwArPlaneType

###### Illustration: Simple summary of the normal vector of a plane, for filtering purposes.

|  |  |
| --- | --- |
| **HwArPlaneType** | **Description** |
| HWAR\_PLANE\_HORIZONTAL\_UPWARD\_FACING | This type means a horizontal plane facing downward. |
| HWAR\_PLANE\_HORIZONTAL\_DOWNWARD\_FACING | This type indicates a horizontal plane facing upward. |
| HWAR\_PLANE\_VERTICAL\_FACING | This type indicates a vertical plane. |
| HWAR\_PLANE\_UNKNOWN\_FACING | This type means this plane is invalid. |

###### HwArPlane

###### Illustration: A detected planar surface (reference type, long-lived).

##### Functions:

###### void HwArPlane\_acquireSubsumedBy(const HwArSession \*session,const HwArPlane \*plane,HwArPlane \*\*out\_subsumed\_by)

###### Illustration: Acquires a reference to the plane subsuming this plane. Two or more planes may be automatically merged into a single parent plane, resulting in this method acquiring the parent plane when called with each child plane.

###### void HwArPlane\_getCenterPose(const HwArSession \*session,const HwArPlane \*plane,HwArPose \*out\_pose)

###### Illustration: Returns the pose of the center of the detected plane. The pose's transformed +Y axis will be point normal out of the plane, with the +X and +Z axes orienting the extents of the bounding rectangle.

###### void HwArPlane\_getExtentX(const HwArSession \*session,const HwArPlane \*plane,float \*out\_extent\_x)

###### Illustration: Returns the length of this HwArPlane's bounding rectangle measured along the local X-axis of the coordinate space centered on the plane.

###### void HwArPlane\_getExtentZ(const HwArSession \*session,const HwArPlane \*plane,float \*out\_extent\_z)

###### Illustration: Returns the length of this HwArPlane's bounding rectangle measured along the local Z-axis of the coordinate space centered on the plane.

###### void HwArPlane\_getPolygon(const HwArSession \*session,const HwArPlane \*plane,float \*out\_polygon\_xz)

###### Illustration: Returns the 2D vertices of a concave polygon approximating the detected plane, in the form [x1, z1, x2, z2, ...]. These X-Z values are in the HwArPlane's local x-z plane (y=0) and must be transformed by the HwArPose to get the boundary in world coordinates. Note that these polygon point are concave.

###### void HwArPlane\_getPolygonSize(const HwArSession \*session,const HwArPlane \*plane,int32\_t \*out\_polygon\_size)

###### Illustration: Retrieves the number of elements (not vertices) in the boundary polygon.

###### The number of vertices is 1/2 this size.

###### void HwArPlane\_getType(const HwArSession \*session,const HwArPlane \*plane,HwArPlaneType \*out\_plane\_type)

###### Illustration: Retrieves the type (orientation) of the plane. See HwArPlaneType.

###### void HwArPlane\_isPoseInExtents(const HwArSession \*session,const HwArPlane \*plane,const HwArPose \*pose,int32\_t \*out\_pose\_in\_extents)

###### Illustration: Sets \*out\_pose\_in\_extents to non-zero if the given pose (usually obtained from a HwArHitResult) is in the plane's rectangular extents.

###### void HwArPlane\_isPoseInPolygon(const HwArSession \*session,const HwArPlane \*plane,const HwArPose \*pose,int32\_t \*out\_pose\_in\_polygon)

###### Illustration: Sets \*out\_pose\_in\_extents to non-zero if the given pose (usually obtained from a HitResult) is in the plane's polygon.

#### HwArPoint

##### Description: Represents a point in space that HUAWEI AR Engine is tracking.

##### Typedefs:

###### HwArPointOrientationMode

###### Illustration: Indicates the orientation mode of the HwArPoint.

|  |  |
| --- | --- |
| **HwArPointOrientationMode** | **Description** |
| HWAR\_POINT\_ORIENTATION\_INITIALIZED\_TO\_IDENTITY | The orientation of the HwArPoint is initialized to identity but may adjust slightly over time. |
| HWAR\_POINT\_ORIENTATION\_ESTIMATED\_SURFACE\_NORMAL | The orientation of the HwArPoint will follow the behavior described in HwArHitResult\_getHitPose(). |

###### HwArPoint

###### Illustration: An arbitrary point in space (reference type, long-lived).

##### Functions:

###### void HwArPoint\_getOrientationMode(const HwArSession \*session,const HwArPoint \*point,HwArPointOrientationMode \*out\_orientation\_mode)

###### Illustration: Returns the OrientationMode of the point. For Point objects created by HwArFrame\_hitTest(). If OrientationMode is HWAR\_POINT\_ORIENTATION\_ESTIMATED\_SURFACE\_NORMAL, then normal of the surface centered around the HwArPoint was estimated succesfully.

###### void HwArPoint\_getPose(const HwArSession \*session,const HwArPoint \*point,HwArPose \*out\_pose)

###### Illustration: Returns the pose of the point.

#### HwArPointCloud

##### Description: Contains a set of observed 3D points and confidence values.

##### Typedefs:

###### HwArPointCloud

###### Illustration: A cloud of tracked 3D visual feature points (reference type, large data).

##### Functions:

###### void HwArPointCloud\_getData(const HwArSession \*session,const HwArPointCloud \*point\_cloud,const float \*\*out\_point\_cloud\_data)

###### Illustration: Retrieves a pointer to the point cloud data. Each point is represented by four consecutive values in the array; first the X, Y, Z position coordinates, followed by a confidence value. This is the same format as described in DEPTH\_POINT\_CLOUD. The pointer returned by this function is valid until HwArPointCloud\_release() is called.

###### void HwArPointCloud\_getNumberOfPoints(const HwArSession \*session,const HwArPointCloud \*point\_cloud,int32\_t \*out\_number\_of\_points)

###### Illustration: Retrieves the number of points in the point cloud.

###### void HwArPointCloud\_getTimestamp(const HwArSession \*session,const HwArPointCloud \*point\_cloud,int64\_t \*out\_timestamp\_ns)

###### Illustration: Returns the timestamp in nanoseconds when this point cloud was observed.This timestamp uses the same time base as HwArFrame\_getTimestamp().

###### void HwArPointCloud\_release(HwArPointCloud \*point\_cloud)

###### Illustration: Releases a reference to the point cloud. This must match a call to HwArFrame\_acquirePointCloud().

#### HwArPose

##### Description: Represents an immutable rigid transformation from one coordinate space to another. As provided from HUAWEI AREngine, HwArPoses always describe the transformation from object's local coordinate space to the world coordinate space. That is, HwArARPoses from HUAWEI AR Engine can be thought of as equivalent to OpenGL model matrices. The transformation is defined using a quaternion rotation about the origin followed by a translation. Coordinate system is right-handled.

##### Typedefs:

###### HwArPose

###### Illustration: A structured rigid transformation (value type).

##### Functions:

###### void HwArPose\_create(const HwArSession \*session,const float \*pose\_raw,HwArPose \*\*out\_pose)

###### Illustration: Allocates and initializes a new pose object. pose\_raw points to an array of 7 floats, describing the rotation (quaternion) and translation of the pose in the same order as the first 7 elements of the Android Sensor.TYPE\_POSE\_6DOF values documented on SensorEvent.values().

###### void HwArPose\_destroy(HwArPose \*pose)

###### Illustration: Releases memory used by a pose object.

###### void HwArPose\_getMatrix(const HwArSession \*session,const HwArPose \*pose,float \*out\_matrix\_col\_major\_4x4)

###### Illustration: Converts a pose into a 4x4 transformation matrix.

###### void HwArPose\_getPoseRaw(const HwArSession \*session,const HwArPose \*pose,float \*out\_pose\_raw)

###### Illustration: Extracts the quaternion rotation and translation from a pose object.

#### HwArTrackable

##### Description: Something that can be tracked and that HwArAnchors can be attached to.

##### Typedefs:

###### HwArTrackableType

###### Illustration: Object types for heterogeneous query/update lists.

|  |  |
| --- | --- |
| **HwArTrackableType** | **Description** |
| HWAR\_TRACKABLE\_BASE\_TRACKABLE | The base Trackable type. |
| HWAR\_TRACKABLE\_NOT\_VALID | An invalid Trackable type. |
| HWAR\_TRACKABLE\_PLANE | The HwArPlane subtype of Trackable. |
| HWAR\_TRACKABLE\_POINT | The Hw[ArPoint](https://developers.google.com/ar/reference/c/group/point.html#group__point_1gab1079c047f9caa38f64c0fb59d3734a1) subtype of Trackable. |

###### HwArTrackable

###### Illustration: Trackable base type (reference type, long-lived).

###### HwArTrackableList

###### Illustration: A list of ArTrackables (value type).

##### Functions:

###### void HwArTrackableList\_acquireItem(const HwArSession \*session,const HwArTrackableList \*trackable\_list,int32\_t index,HwArTrackable \*\*out\_trackable)

###### Illustration: Acquires a reference to an indexed entry in the list. This call must eventually be matched with a call to HwArTrackable\_release().

###### void HwArTrackableList\_create(const HwArSession \*session,HwArTrackableList \*\*out\_trackable\_list)

###### Illustration: Creates a trackable list object.

###### void HwArTrackableList\_destroy(HwArTrackableList \*trackable\_list)

###### Illustration: Releases the memory used by a trackable list object, along with all the anchor references it holds.

###### void HwArTrackableList\_getSize(const HwArSession \*session,const HwArTrackableList \*trackable\_list,int32\_t \*out\_size)

###### Illustration: Retrieves the number of trackables in this list.

###### HwArStatus HwArTrackable\_acquireNewAnchor(HwArSession \*session,HwArTrackable \*trackable,HwArPose \*pose,HwArAnchor \*\*out\_anchor)

###### Illustration: Creates an HwAnchor at the given pose in the world coordinate space, attached to this Trackable, and acquires a reference to it.

###### void HwArTrackable\_getAnchors(const HwArSession \*session,const HwArTrackable \*trackable,HwArAnchorList \*out\_anchor\_list)

###### Illustration: Gets the set of HwAranchors attached to this trackable.

###### void HwArTrackable\_getTrackingState(const HwArSession \*session,const HwArTrackable \*trackable,HwArTrackingState \*out\_tracking\_state)

###### Illustration: Retrieves the current state of HUAWEI AR Engine's knowledge of the pose of this trackable.

###### void HwArTrackable\_getType(const HwArSession \*session,const HwArTrackable \*trackable,HwArTrackableType \*out\_trackable\_type)

###### Illustration: Retrieves the type of the trackable. See ArTrackableType for valid types.

###### void HwArTrackable\_release(HwArTrackable \*trackable)

###### Illustration: Releases a reference to a trackable.

#### HwArEnginesSelector

##### Description: Used to query and set the engine supported by this device.

##### Typedefs:

###### HwArEnginesType

###### Illustration: The type of engine.

|  |  |
| --- | --- |
| **HwArEnginesType** | **Description** |
| HWAR\_NONE | No engine, this value cannot be used to set the engine, only as a return value. |
| HWAR\_ENGINE | Huawei AREngine. |
| GOOGLE\_ARCORE | Google ARCore. |

###### HwArEnginesAvaliblity

###### Illustration: the type of engines that is available for the device.

|  |  |
| --- | --- |
| **HwArEnginesAvaliblity** | **Description** |
| HWAR\_NONE\_ENGINES\_SUPPORTED | The device does not support any engine. |
| HWAR\_ENGINE\_SUPPORTED | Only Huawei AREngine is supported. |
| GOOGLE\_ARCORE\_SUPPORTED | Only Google ARCore is supported. |
| HWAR\_BOTH\_ENGINES\_SUPPORTED | Support both Huawei AREngine and Google ARCore. |

##### Functions:

###### HwArEnginesAvaliblity HwArEnginesSelector\_checkAllAvailableEngines(void \*env,void \*application\_context)

###### Illustration: Check the type of AR engines supported by the device and perform default initialization based on the type supported by the engine. If the HwArEnginesAvaliblity is only one engine, the engine is initialized. If the dual engine is supported, the Huawei self-research engine HUAWEI AR Engine is initialized. If no engine supports it, an exception is thrown and the program ends.

###### Note: Since the google play store is not available in China, if you want to check the support of Google ARCore, please add ArCore 1.2 dependencies in dependencies and set ARCore to required in AndroidManifest.xml.

###### void HwArEnginesSelector\_setAREngine(HwArEnginesType ar\_options,HwArEnginesAvaliblity enginesavailabile)

###### Illustration: set the initialized engine type based on the result of HwArEnginesSelector\_checkAllAvailableEngines.

###### HwArEnginesType HwArEnginesSelector\_getCreatedEngine()

###### Illustration: Get the type of engine currently in use. Returns HWAR\_NONE if the engine has not been initialized before the function is called.

#### HwArHand

##### Description: Return tracking results for hand tracking, including hand skeleton data and gesture recognition results，derived from ARTrackableBase.

##### Typedefs:

###### HwArHand

###### Illustration: A hand object attached to a trackable.

##### Functions:

###### void HwArHand\_getGestureCoordinateSystem(const HwArSession \*session,const HwArHand \*hand, int32\_t\* gestureCoordinateSystem)

###### Illustration: Get the coordinate system.

|  |  |
| --- | --- |
| **CoordinateSystemType** | **Description** |
| COORDINATE\_SYSTEM\_TYPE\_2D\_IMAGE | 3D world coordinate system |
| COORDINATE\_SYSTEM\_TYPE\_3D\_WORLD | 2D image coordinate system |
| COORDINATE\_SYSTEM\_TYPE\_3D\_SELF | 3D coordinate system of the object itself |

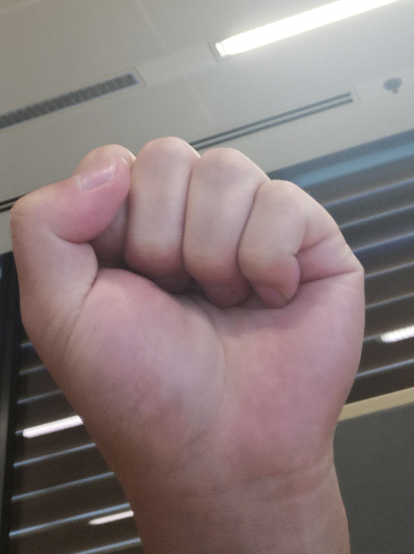
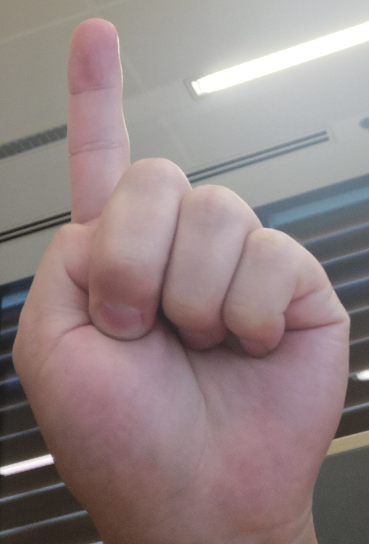
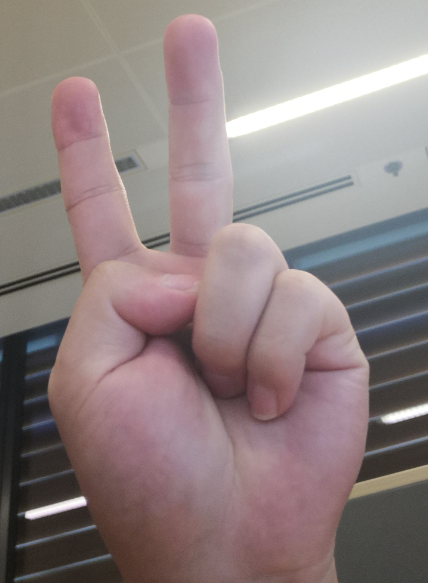
###### void HwArHand\_getHandId(const HwArSession \*session,const HwArHand \*hand, int32\_t\* handId)

###### Illustration: Get the hand ID**.**

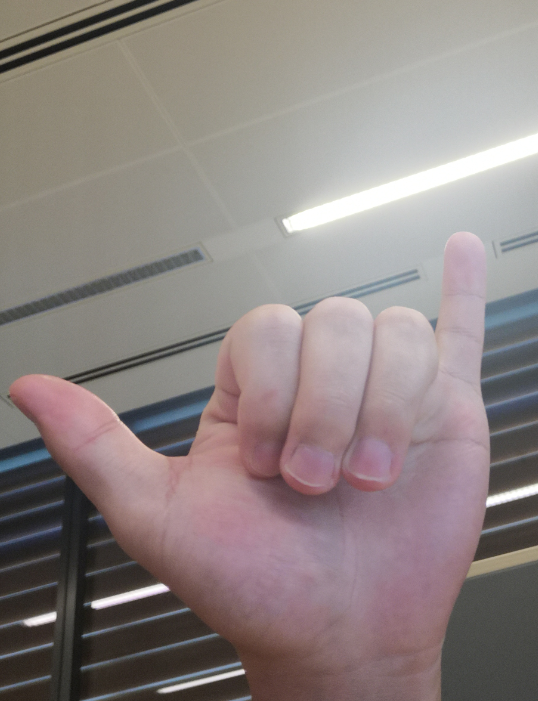
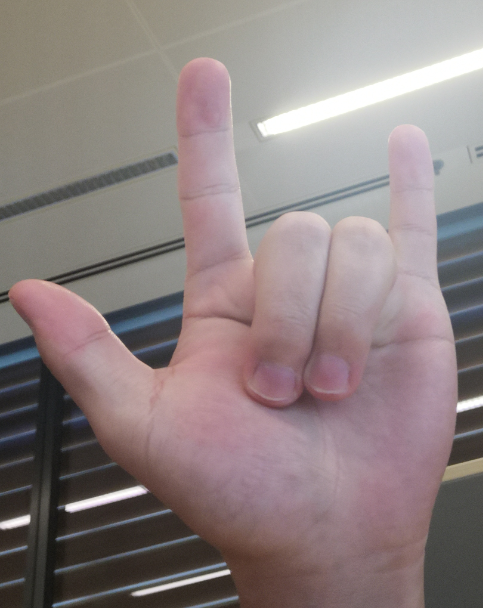
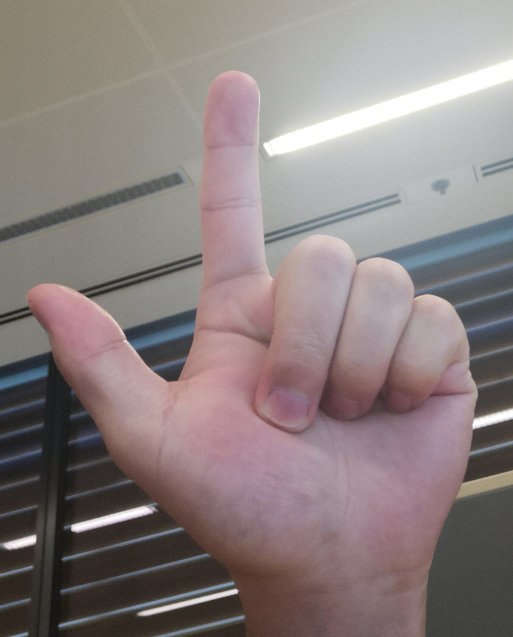
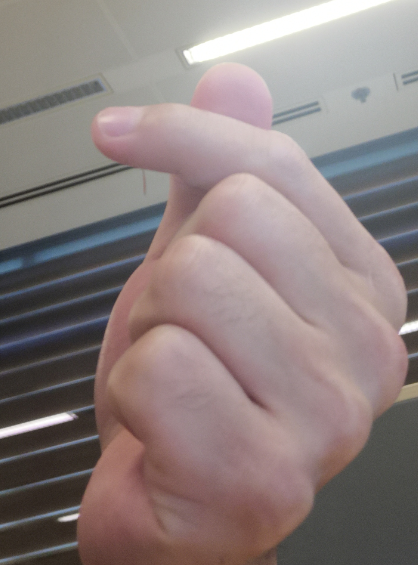
###### void HwArHand\_getGestureType(const HwArSession \*session,const HwArHand \*hand, int32\_t\* gestureType)

###### Illustration: Get the gesture type, the value is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **GestureType** | **Value** | **Support in disable depth** | **Support in enable depth** |
| Gesture 0（Make a fist） | 0 | Y | Y |
| Gesture 1（Erect index finger or middle finger） | 1 | Y | N |
| Gesture 2（Erect index finger and middle finger） | 2 | N | Y |
| Gesture 5（Five fingers open） | 5 | Y | Y |
| Gesture 6（Make a telephone call） | 6 | Y | Y |
| Gesture 7（I love you hand gesture） | 7 | Y | Y |
| Gesture 8（Erect thumb and middle finger） | 8 | N | Y |
| Gesture 10（Finger Heart） | 10 | Y | N |
| Other gestures | -1 | -- | -- |

Gesture0 Gesture 1 Gesture 2 Gesture 5

Gesture 6 Gesture 7 Gesture 8 Gesture 10

###### void HwArHand\_getHandType(const HwArSession \*session,const HwArHand \*hand, int32\_t\* handType)

###### Illustration: Get the hand type，0 is the right hand, and 1 is the left hand.

###### void HwArHand\_getGestureHandBox(const HwArSession \*session,const HwArHand \*hand, const float \*\*vct)

###### Illustration: Get the Vertex coordinates of a box which indicating the position of hand, based on the OpenGL coordinate system.

###### void HwArHand\_getGestureCenter(const HwArSession \*session,const HwArHand \*hand, const float \*\*vct)

###### Illustration: Get the coordinates of the center point of the hand.

###### void HwArHand\_getGestureActionSize(const HwArSession \*session, const HwArHand \*hand, int32\_t \*count)

###### Illustration: Retrieves the number of gestures.

###### void HwArHand\_getGestureAction(const HwArSession \*session,const HwArHand \*hand, const int32\_t \*\*vct)

###### Illustration: Get gestures (for example, left waving and right waving).

###### void HwArHand\_getGestureOrientation(const HwArSession \*session, const HwArHand \*hand, const float \*\*vct)

###### Illustration: Get the gesture orientations.

###### void HwArHand\_getSkeletonCoordinateSystem(const HwArSession \*session, const HwArHand \*hand, int32\_t\* skeletonCoordinateSystem)

###### Illustration: Returns the coordinate system for hand skeletons.

###### void HwArHand\_getHandSkeletonType(const HwArSession \*session,const HwArHand \*hand, int32\_t\* handType)

###### Illustration: Get the type of hand skeleton.

###### void HwArHand\_getHandSkeletonCount(const HwArSession \*session,const HwArHand \*hand, int32\_t \*count)

###### Illustration: Retrieves the number of hand skeleton.

###### void HwArHand\_getHandSkeletonArray(const HwArSession \*session,const HwArHand \*hand, const float \*\*vct)

###### Illustration: Returns the coordinates of the hand skeleton points.

###### void HwArHand\_getHandSkeletonConnectionSize(const HwArSession \*session,const HwArHand \*hand, int32\_t \*count)

###### Illustration: Retrieves the number of hand skeleton Connections.

###### void HwArHand\_getHandSkeletonConnection(const HwArSession \*session,const HwArHand \*hand, const int32\_t \*\*vct)

###### Illustration: Get the data of human skeleton point connection relationship, the data format is:[p0,p1;p0,p3;p0,p5;p1,p2;…].

#### HwArBody

##### Description: Return tracking results for body skeleton tracking，include body skeleton data，derived from ARTrackableBase.

##### Typedefs:

###### HwArBody

###### Illustration: a body object attached to a trackable.

##### Functions:

###### void HwArBody\_getCoordinateSyetemType(const HwArSession \*session,const HwArBody \*body, int32\_t \*out\_coordinateSystemType)

###### Illustration: obtain the coordinate system type.

|  |  |
| --- | --- |
| **HwArCoordinateSystemType** | **Description** |
| COORDINATE\_SYSTEM\_TYPE\_3D\_WORLD | 3D world coordinate system |
| COORDINATE\_SYSTEM\_TYPE\_3D\_SELF | 3D coordinate system of the object itself |
| COORDINATE\_SYSTEM\_TYPE\_2D\_IMAGE | 2D image coordinate system |

###### void HwArBody\_getBodyId(const HwArSession \*session,const HwArBody \*body, int32\_t \*out\_personId)

###### Illustration: Returns the body ID.

###### void HwArBody\_getSkeletonPointCount(const HwArSession \*session,const HwArBody \*body, int32\_t \*out\_pointCount)

###### Illustration: Retrieves the number of body skeleton point.

###### void HwArBody\_getSkeletonTypes(const HwArSession \*session,const HwArBody \*body, const HwArBodySkeletonType \*\*out\_skeleton\_types)

###### Illustration: get the skeletion types.

###### void HwArBody\_getSkeletonPointIsExist2D(const HwArSession \*session,const HwArBody \*body, const int32\_t \*\*out\_skeletonPintIsExist\_2D)

###### Illustration: Gets the existences of human body skeleton points，the data format is:[0,1,0,0…].

###### void HwArBody\_getSkeletonPointIsExist3D(const HwArSession \*session,const HwArBody \*body, const int32\_t \*\*out\_skeletonPintIsExist\_3D)

###### Illustration: Gets the existences of human body skeleton points，the data format is:[0,1,0,0…].

###### void HwArBody\_getSkeletonPoint2D(const HwArSession \*session,const HwArBody \*body, const float \*\*out\_point2D)

###### Illustration: Gets the 2D coordinates of the human body skeleton points,the data format is :[x0,y0,x1,y1,…].

###### void HwArBody\_getSkeletonPoint3D(const HwArSession \*session, const HwArBody \*body, const float \*\*out\_point3D)

###### Illustration: Gets the 3D coordinates of the human body skeleton points, the data format is :[x0,y0,z0,x1,y1,z1…].

###### void HwArBody\_getSkeletonConnectionCount(const HwArSession \*session, const HwArBody \*body, int32\_t \*out\_connetionCount)

###### Illustration: Retrieves the number of body skeleton connections.

###### void HwArBody\_getSkeletonConnection(const HwArSession \*session,const HwArBody \*body, const int32\_t \*\*out\_skeletonConnection)

###### Illustration: Gets the body skeleton connection data，the sequence of the connection data is:[p0,p1;p0,p3;p0,p5;p1,p2;…].

###### void HwArBody\_getSkeletonConfidence(const HwArSession \*session, const HwArBody \*body, const float \*\*out\_confidence)

###### Illustration:get every confidence of Skeletons,the value is from 0 to 1.

###### void HwArBody\_getBodyAction(const HwArSession \*session, const HwArBody \*body, HwArBodyAction \*out\_bodyAction)

###### Illustration: Gets body actions.

###### void HwArBody\_getMaskConfidence(const HwArSession \*session, const HwArBody \*body,const float \*\*out\_confidence)

###### Illustration: get every confidence of Mask Body,array size is TextureDimension\_width\* TextureDimension\_height, the value is from 0 to 1.

###### void HwArBody\_getMaskDepth(const HwArSession \*session, const HwArBody \*body,const short \*\*out\_depth)

###### Illustration: get every depth of Mask Body,array size is TextureDimension\_width\* TextureDimension\_height,the format of each value is depth16,refer to: https://developer.android.com/reference/android/graphics/ImageFormat#DEPTH16

#### HwArEnginesApk

##### Description: used to manage the status of ARCore server and HUAWEI AR Engine server.

##### Typedefs:

###### HwArAvailability

###### Illustration: Describes the current state of service availability on the device.

|  |  |
| --- | --- |
| **HwArAvailability** | **Description** |
| HWAR\_AVAILABILITY\_UNKNOWN\_ERROR | An internal error occurred while determining service availability. |
| HWAR\_AVAILABILITY\_UNKNOWN\_CHECKING | Service is not installed, and a query has been issued to check if service is supported. |
| HWAR\_AVAILABILITY\_UNKNOWN\_TIMED\_OUT | Service is not installed, and the query to check if service is supported timed out. This may be due to the device being offline. |
| HWAR\_AVAILABILITY\_UNSUPPORTED\_DEVICE\_NOT\_CAPABLE | Service is not supported on this device. |
| HWAR\_AVAILABILITY\_SUPPORTED\_NOT\_INSTALLED | The device is supported, but the Service APK is not installed. |
| HWAR\_AVAILABILITY\_SUPPORTED\_APK\_TOO\_OLD | The device is supported, and a version of the service APK is installed, but that service APK version is too old. |
| HWAR\_AVAILABILITY\_SUPPORTED\_INSTALLED | Service is supported, installed, and available to use. |

###### HwArInstallStatus

###### Illustration: Indicates the outcome of a call to HwArApk\_requestInstall().

|  |  |
| --- | --- |
| **HwArInstallStatus** | **Description** |
| HWAR\_INSTALL\_STATUS\_INSTALLED | The requested resource is already installed. |
| HWAR\_INSTALL\_STATUS\_INSTALL\_REQUESTED | Installation of the resource was requested. The current activity will be paused. |

###### HwArInstallBehavior (Not implemented)

###### Illustration: Controls the behavior of the installation UI.

|  |  |
| --- | --- |
| **HwArInstallBehavior** | **Description** |
| HWAR\_INSTALL\_BEHAVIOR\_REQUIRED | Hide the Cancel button during initial prompt and prevent user from exiting via tap-outside. |
| HWAR\_INSTALL\_BEHAVIOR\_OPTIONAL | Include Cancel button in initial prompt and allow easily backing out after installation has been initiated. |

##### Functions:

###### void HwArEnginesApk\_checkAvailability(void \*env, void \*application\_context, HwArAvailability \*out\_availability)

###### Illustration: Check if the device supports the currently set engine (set by the HwArEnginesSelector\_setAREngine() method).

###### HwArStatus HwArEnginesApk\_requestInstall(void \*env, void \*application\_activity, bool user\_requested\_install, HwArInstallStatus \*out\_install\_status)

###### Illustration: Initiates installation of service if needed. When your apllication launches or enters an AR mode, it should call this method with user\_requested\_install = 1.

#### HwArConfiguration

##### Description: To configure the HUAWEI AR Engine system，including the three modes: LightingMode, PlaneFindingMode, UpdateMode, CameraLensFacing and HwArType.

##### Typedefs:

###### HwArLightEstimationMode

###### Illustration: Select the behavior of the lighting estimation subsystem.

|  |  |
| --- | --- |
| **HwArLightEstimationMode** | **Description** |
| HWAR\_LIGHT\_ESTIMATION\_MODE\_AMBIENT\_INTENSITY | Lighting estimation is enabled, generating a single-value intensity estimate. |
| HWAR\_LIGHT\_ESTIMATION\_MODE\_DISABLED | Lighting estimation is disabled. |

###### HwArPlaneFindingMode

###### Illustration: Select the behavior of the plane detection subsystem.

|  |  |
| --- | --- |
| **HwArPlaneFindingMode** | **Description** |
| HWAR\_PLANE\_FINDING\_MODE\_DISABLED | Plane detection is disabled. |
| HWAR\_PLANE\_FINDING\_MODE\_HORIZONTAL | Detection of only horizontal planes is enabled. |
| HWAR\_PLANE\_FINDING\_MODE\_VERTICAL | Detection of horizontal and vertical planes is enabled. |
| HWAR\_PLANE\_FINDING\_MODE\_HORIZONTAL\_AND\_VERTICAL | Detection of only vertical planes is enabled. |

###### HwArUpdateMode

###### Illustration: Selects the behavior of HwArSession\_update().

|  |  |
| --- | --- |
| **HwArUpdateMode** | **Description** |
| HWAR\_UPDATE\_MODE\_BLOCKING | In BLOCKING model, update() will wait until a new camera image is available. |
| HWAR\_UPDATE\_MODE\_LATEST\_CAMERA\_IMAGE | In LATEST\_CAMERA\_IMAGE model, update() will return immediately without blocking. If no new camera image is available, then it will return the most recent HwArFrame. |

###### HwArType

###### Illustration: HUAWEI AR Engine supports different modes of AR. Currently only one: HWAR\_TYPE\_WORLD is the SLAM-capable AR capability.

###### HwArCameraLensFacing

###### Illustration: Describe the settings for the front and rear cameras.

|  |  |
| --- | --- |
| **HwArCameraLensFacing** | **Description** |
| HWAR\_CAMERA\_FACING\_REAR | Set to rear camera. |
| HWAR\_CAMERA\_FACING\_FRONT | Set to front camera. |

###### HwArPowerMode

###### Illustration: Describe the settings of power mode.

|  |  |
| --- | --- |
| **PowerMode** | **Description** |
| HWAR\_POWER\_MODE\_NORMAL | Normal mode |
| HWAR\_POWER\_MODE\_POWER\_SAVING | Power saving mode |
| HWAR\_POWER\_MODE\_ULTRA\_POWER\_SAVING | Ultra Power Saving Mode |

###### HwArFocusMode

###### Illustration: Describe the settings of focus mode.

|  |  |
| --- | --- |
| **FocusMode** | **Description** |
| HWAR\_FOCUS\_MODE\_FIXED | Fixed focus mode |
| HWAR\_FOCUS\_MODE\_AUTO | Auto focus mode |

###### HwArConfig

###### Illustration: An opaque session configuration object (value type).

##### Functions:

###### void HwArConfig\_create(const HwArSession \*session,HwArConfig \*\*out\_config)

###### Illustration: Creates a new configuration object and initializes it to a sensible default configuration.

###### void HwArConfig\_destroy(HwArConfig \*config)

###### Illustration: Releases memory used by the provided configuration object.

###### void HwArConfig\_setArType(const HwArSession \*session,HwArConfig \*config, HwArType type)

###### Illustration: Set the type of the AR mode.

###### void HwArConfig\_getArType(const HwArSession \*session, const HwArConfig \*config,

###### HwArType \*type)

###### Illustration: get the type of the AR mode.

###### void HwArConfig\_setCameraLensFacing(const HwArSession \*session, HwArConfig \*config, HwArCameraLensFacing facing)

###### Illustration: Set the orientation of the camera.

###### void HwArConfig\_getCameraLensFacing(const HwArSession \*session, const HwArConfig \*config, HwArCameraLensFacing \*facing)

###### Illustration: Get the orientation of the camera.

###### void HwArConfig\_getLightEstimationMode(const HwArSession \*session,const HwArConfig \*config,HwArLightEstimationMode \*light\_estimation\_mode)

###### Illustration: Stores the currently configured lighting estimation mode into \*light\_estimation\_mode.

###### void HwArConfig\_setLightEstimationMode(const HwArSession \*session,HwArConfig \*config,HwArLightEstimationMode light\_estimation\_mode)

###### Illustration: Sets the lighting estimation mode that should be used. See ArLightEstimationMode for available options.

###### void HwArConfig\_getPlaneFindingMode(const HwArSession \*session,const HwArConfig \*config,HwArPlaneFindingMode \*plane\_finding\_mode)

###### Illustration: Stores the currently configured plane finding mode into \*plane\_finding\_mode.

###### void HwArConfig\_setPlaneFindingMode(const HwArSession \*session,HwArConfig \*config,HwArPlaneFindingMode plane\_finding\_mode)

###### Illustration: Sets the plane finding mode that should be used. See HwArPlaneFindingMode for available options.

###### void HwArConfig\_getUpdateMode(const HwArSession \*session,const HwArConfig \*config,HwArUpdateMode \*update\_mode)

###### Illustration: Stores the currently configured behavior of HwArSession\_update() into \*update\_mode.

###### void HwArConfig\_setUpdateMode(const HwArSession \*session,HwArConfig \*config,HwArUpdateMode update\_mode) public void setLightingMode(LightingMode lightingMode)

###### Illustration: Sets the behavior of HwArSession\_update(). See HwArUpdateMode for available options.

###### void HwArConfig\_getAugmentedImageDatabase(const HwArSession \*session, const HwArConfig \*config, HwArAugmentedImageDatabase \*out\_augmented\_image\_database);

###### Illustration: Return the setted AugmentedImageDatabase.

###### void HwArConfig\_getEnableItem (const HwArSession\* session, const HwArConfig\* config, uint64\_t \* item);

###### Illustration:Return the capability of the camera.

###### void HwArConfig\_getPowerMode(const HwArSession \*session, const HwArConfig \*config, HwArPowerMode \*power\_mode);

###### Illustration: Return the power mode.

###### void HwArConfig\_setAugmentedImageDatabase(const HwArSession \*session, HwArConfig \*config,const HwArAugmentedImageDatabase \*augmented\_image\_database);

###### Illustration:set the capability of the camera.

###### void HwArConfig\_setEnableItem(const HwArSession\* session, HwArConfig\* config, uint64\_t \* item);

###### Illustration:set the capability of the camera.

|  |  |
| --- | --- |
| **EnableItem** | **Description** |
| ENABLE\_NULL = 0 | None |
| ENABLE\_DEPTH = 1 << 0 | Enable depth capability, enable by default |
| ENABLE\_MASK = 1 << 1 | Enable Mask capability |

###### void HwArConfig\_setPowerMode(const HwArSession \*session, HwArConfig \*config, HwArPowerMode power\_mode);

###### Illustration:Set the power mode.

###### void HwArConfig\_getFocusMode(const HwArSession \*session, const HwArConfig \*config, HwArFocusMode \*focus\_mode);

###### Illustration:Return the focus mode.

###### void HwArConfig\_setFocusMode(const HwArSession \*session, HwArConfig \*config,HwArFocusMode focus\_mode);

###### Illustration:Set the focus mode.

#### HwArSession

##### Description: Manages HUAWEI AR Engine system state. This class is the main entry point to HUAWEI AR Engine. This class allows the user to create a session, configure it, start/stop it, and most importantly receive frames that can access to camera image and device pose.

##### Typedefs:

###### HwArSession

###### Illustration: The HUAWEI AR Engine session (value type).

##### Functions:

###### HwArStatus HwArSession\_acquireNewAnchor(HwArSession \*session,const HwArPose \*pose,HwArAnchor \*\*out\_anchor)

###### Illustration: Defines a tracked location in the physical world.

###### HwArStatus HwArSession\_checkSupported(const HwArSession \*session,const HwArConfig \*config)

###### Illustration: Checks if the provided configuration is usable on the device. Note: This methods only returns true currently.

###### HwArStatus HwArSession\_configure(HwArSession \*session,const HwArConfig \*config)

###### Illustration: Configures the session with the given config. Note: a session is always initially configured with the default config. This should be called if a configuration different than default is needed.

###### HwArStatus HwArSession\_create(void \*env,void \*application\_context,HwArSession \*\*out\_session\_pointer)

###### Illustration: Attempts to create a new HUAWEI AR Engine session. This is the entry point of HUAWEI AR Engine. This function MUST be the first HUAWEI AR Engine call made by an application.

###### void HwArSession\_destroy(HwArSession \*session)

###### Illustration: Releases resources used by an HUAWEI AR Engine session. This method will take several seconds to complete.

###### void HwArSession\_getAllAnchors(const HwArSession \*session,HwArAnchorList \*out\_anchor\_list)

###### Illustration: Returns all known anchors, including those not currently tracked.

###### void HwArSession\_getAllTrackables(const HwArSession \*session,HwArTrackableType filter\_type,HwArTrackableList \*out\_trackable\_list)

###### Illustration: Returns the list of all known trackables.

###### HwArStatus HwArSession\_pause(HwArSession \*session)

###### Illustration: Pause the current session. This method will stop the camera feed and release resources. The session can be restarted again by calling HwArSession\_resume().

###### HwArStatus HwArSession\_resume(HwArSession \*session)

###### Illustration: Starts or resumes the HUAWEI AR Engine Session. Typically this should be called from Activity.onResume() .

###### void HwArSession\_setCameraTextureName(HwArSession \*session,uint32\_t texture\_id)

###### Illustration: Sets the OpenGL texture name (id) that will allow GPU access to the camera image. The provided ID should have been created with glGenTextures(). The resulting texture must be bound to the GL\_TEXTURE\_EXTERNAL\_OES target for use. Shaders accessing this texture must use a samplerExternalOES sampler. See sample code for an example.

###### void HwArSession\_stop(HwArSession \*session)

###### Illustration: Stop the current session. This method will stop the camera feed and release resources. To restart, a new session is needed.

###### void HwArSession\_setDisplayGeometry(HwArSession \*session,int32\_t rotation,int32\_t width,int32\_t height)

###### Illustration: Sets the aspect ratio, coordinate scaling, and display rotation. This data is used by UV conversion, projection matrix generation, and hit test logic.

###### HwArStatus HwArSession\_update(HwArSession \*session,HwArFrame \*out\_frame)

###### Illustration: Updates the state of the HUAWEI AR Engine system. This includes: receiving a new camera frame, updating the location of the device, updating the location of tracking anchors, updating detected planes, etc. If the ARConfig.UpdateMode is BLOCKING，this method will be blocked until a new camera image is available.

###### void HwArSession\_getCameraConfig(const HwArSession \*session, HwArCameraConfig \*out\_camera\_config)

###### Illustration: Gets the ArCameraConfig that the ArSession is currently using

#### HwArCamera

##### Description: Provides information about the camera that is used to capture images.

##### Typedefs:

###### HwArCamera

###### The virtual and physical camera (reference type, long-lived).

##### Functions:

###### void HwArCamera\_getDisplayOrientedPose(const HwArSession \*session,const HwArCamera \*camera,HwArPose \*out\_pose)

###### Illustration: Sets out\_pose to the pose of the user's device in the world coordinate space at the time of capture of the current camera texture. The position of the pose is located at the device's camera, while the orientation approximately matches the orientation of the display (considering display rotation), using OpenGL camera conventions (+X right, +Y up, -Z in the direction the camera is looking).

###### void HwArCamera\_getPose(const HwArSession \*session,const HwArCamera \*camera,HwArPose \*out\_pose)

###### Illustration: Sets out\_pose to the pose of the user's device in the world coordinate space at the time of capture of the current camera texture. The position and orientation of the pose follow the device's physical camera (they are not affected by display orientation) and uses OpenGL camera conventions (+X right, +Y up, -Z in the direction the camera is looking).

###### void HwArCamera\_getProjectionMatrix(const HwArSession \*session,const HwArCamera \*camera,float near,float far,float \*dest\_col\_major\_4x4)

###### Illustration: Computes a projection matrix for rendering virtual content on top of the camera image.

###### void HwArCamera\_getTrackingState(const HwArSession \*session,const HwArCamera \*camera,HwArTrackingState \*out\_tracking\_state)

###### Illustration: Gets the current state of the pose of this camera. If this state is anything other than HWAR\_TRACKING\_STATE\_TRACKING the Camera's pose should not be considered useful.

###### void HwArCamera\_getViewMatrix(const HwArSession \*session,const HwArCamera \*camera,float \*out\_col\_major\_4x4)

###### Illustration: Returns the view matrix for the camera for this frame.

###### void HwArCamera\_release(HwArCamera \*camera)

###### Illustration: Releases a reference to the camera.

#### HwArCameraConfig

##### Description: Camera configuration

##### Typedefs：

###### HwArCameraConfig

###### Illustration: A camera config struct that contains the config supported by the physical camera obtained from the low level device profiles.

##### Functions：

###### void HwArCameraConfig\_getImageDimensions(const HwArSession \*session, const HwArCameraConfig \*camera\_config, int32\_t \*out\_width, int32\_t \*out\_height)

###### Illustration: Obtains the camera image dimensions for the given camera config.

###### void HwArCameraConfig\_getTextureDimensions(const HwArSession \*session, const HwArCameraConfig \*camera\_config, int32\_t \*out\_width, int32\_t \*out\_height)

###### Illustration: Obtains the texture dimensions for the given camera config.

#### HwArFrame

##### Description: Per-frame state.

##### Typedefs:

###### HwArFrame

###### Illustration: The world state resulting from an update (value type).

##### Functions:

###### void HwArFrame\_acquireCamera(const HwArSession \*session,const HwArFrame \*frame,HwArCamera \*\*out\_camera)

###### Illustration: Returns the camera object for the session.

###### HwArStatus HwArFrame\_acquireImageMetadata(const HwArSession \*session,const HwArFrame \*frame,HwArImageMetadata \*\*out\_metadata)

###### Illustration: Gets the camera metadata for the current camera image

###### HwArStatus HwArFrame\_acquirePointCloud(const HwArSession \*session,const HwArFrame \*frame,HwArPointCloud \*\*out\_point\_cloud)

###### Illustration: Acquires the current set of estimated 3d points attached to real-world geometry.

###### void HwArFrame\_create(const HwArSession \*session,HwArFrame \*\*out\_frame)

###### Illustration: Allocates a new ArFrame object, storing the pointer into \*out\_frame. Note: the same HwArFrame can be used repeatedly when calling HwArSession\_update.

###### void HwArFrame\_destroy(HwArFrame \*frame)

###### Illustration: Releases an ArFrame and any references it holds.

###### void HwArFrame\_getDisplayGeometryChanged(const HwArSession \*session,const HwArFrame \*frame,int32\_t \*out\_geometry\_changed)

###### Illustration: Checks if the display rotation or viewport geometry changed since the previous Frame. The application should re-query HwArCamera\_getProjectionMatrix() and HwArFrame\_transformDisplayUvCoords() whenever this emits non-zero.

###### void HwArFrame\_getLightEstimate(const HwArSession \*session,const HwArFrame \*frame,HwArLightEstimate \*out\_light\_estimate)

###### Illustration: Gets the current ambient light estimate, if light estimation was enabled. void HwArFrame\_getTimestamp(const HwArSession \*session,const HwArFrame \*frame,int64\_t \*out\_timestamp\_ns)

###### Illustration: Returns the timestamp in nanoseconds when this image was captured.

###### void HwArFrame\_getUpdatedAnchors(const HwArSession \*session,const HwArFrame \*frame,HwArAnchorList \*out\_anchor\_list)

###### Illustration: Gets the set of anchors that were changed by the HwArSession\_update() that produced this Frame.

###### void HwArFrame\_getUpdatedTrackables(const HwArSession \*session,const HwArFrame \*frame,HwArTrackableType filter\_type,HwArTrackableList \*out\_trackable\_list)

###### Illustration: Gets the set of trackables of a particular type that were changed by the ArSession\_update() call that produced this Frame.

###### void HwArFrame\_hitTest(const HwArSession \*session,const HwArFrame \*frame,float pixel\_x,float pixel\_y,HwArHitResultList \*hit\_result\_list)

###### Illustration: Performs a ray cast from the user's device in the direction of the given location in the camera view. Intersections with detected scene geometry are returned, sorted by distance from the device; the nearest intersection is returned first. *xPX* is the x coordinate in pixels, and *yPX* is the y coordinate in pixels.

###### void HwArFrame\_transformDisplayUvCoords(const HwArSession \*session,const HwArFrame \*frame,int32\_t num\_elements,const float \*uvs\_in,float \*uvs\_out)

###### Illustration: Transform the given texture coordinates to correctly show the background image. This will account for the display rotation, and any additional required adjustment. For performance, this function should be called only if HwArFrame\_hasDisplayGeometryChanged() emits true.

#### HwArHitResult

##### Description: Defines an intersection between a ray and estimated real-world geometry.

##### Typedefs:

###### HwArHitResult

###### Illustration: A single trackable hit (value type).

###### HwArHitResultList

###### Illustration: A list of hit test results (value type).

##### Functions:

###### void HwArHitResultList\_create(const HwArSession \*session,HwArHitResultList \*\*out\_hit\_result\_list)

###### Illustration: Creates a hit result list object.

###### void HwArHitResultList\_destroy(HwArHitResultList \*hit\_result\_list)

###### Illustration: Releases the memory used by a hit result list object, along with all the trackable references it holds.

###### void HwArHitResultList\_getItem(const HwArSession \*session,const HwArHitResultList \*hit\_result\_list,int32\_t index,HwArHitResult \*out\_hit\_result)

###### Illustration: Copies an indexed entry in the list. This acquires a reference to any trackable referenced by the item, and releases any reference currently held by the provided result object.

###### void HwArHitResultList\_getSize(const HwArSession \*session,const HwArHitResultList \*hit\_result\_list,int32\_t \*out\_size)

###### Illustration: Retrieves the number of hit results in this list.

###### HwArStatus HwArHitResult\_acquireNewAnchor(HwArSession \*session,HwArHitResult \*hit\_result,HwArAnchor \*\*out\_anchor)

###### Illustration: Creates a new anchor at the hit location.

###### void HwArHitResult\_acquireTrackable(const HwArSession \*session,const HwArHitResult \*hit\_result,HwArTrackable \*\*out\_trackable)

###### Illustration: Acquires reference to the hit trackable.

###### void HwArHitResult\_create(const HwArSession \*session,HwArHitResult \*\*out\_hit\_result)

###### Illustration: Allocates an empty hit result object.

###### void HwArHitResult\_destroy(HwArHitResult \*hit\_result)

###### Illustration: Releases the memory used by a hit result object, along with any trackable reference it holds.

###### void HwArHitResult\_getDistance(const HwArSession \*session,const HwArHitResult \*hit\_result,float \*out\_distance)

###### Illustration: Returns the distance from the camera to the hit location, in meters.

###### void HwArHitResult\_getHitPose(const HwArSession \*session,const HwArHitResult \*hit\_result,HwArPose \*out\_pose)

###### Illustration: Returns the pose of the intersection between a ray and detected real-world geometry. The position is the location in space where the ray intersected the geometry. The orientation’s definition differs depending on the object that was hit. When the ray hit a HwArPlane, the local coordinate space is: X+ is perpendicular to the cast ray and parallel to the plane, Y+ points along the plane normal (up, for HWAR\_PLANE\_HORIZONTAL\_UPWARD\_FACING planes), and Z+ is parallel to the plane, pointing roughly toward the user's device. When it hit a point, the coordinate becomes: X+ is perpendicular to the cast ray and points right from the perspective of the user's device, Y+ points up, and Z+ points roughly toward the user's device.