|  |  |
| --- | --- |
| 产品名称InputSystem |  |
|  |  |
| 产品版本InputSystem | Total 21 pages 共 21页 |
|  |

InputSystem Release Notes

|  |  |  |  |
| --- | --- | --- | --- |
| Prepared by  拟制 | Unity内容组 | Date  日期 | 2020/01/02 |
| Reviewed by  审核 |  | Date  日期 |  |
| Approved by  批准 |  | Date  日期 |  |



Shadow Creator Information Technology Co.,Ltd.

上海影创信息科技有限公司

All rights reserved

版权所有 侵权必究

Revision Record 修订记录

| Date  日期 | Revision Version  修订 版本 | Release Notes | Change Description  修改描述 | Author  作者 |
| --- | --- | --- | --- | --- |
| 2020/01/02 | V1.0.0 | InputSystem | 初稿完成 | 王超群 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Catalog 目 录

1 InputSystem简介 4

1.1 简介 4

1.2 UML图 4

1.3 概念 5

1.4 输入设备基类简介 5

2 输入设备种类简介 6

2.1 Head输入设备 6

2.2 HandShank输入设备 7

2.3 Gesture26Dof输入设备 7

3 输入设备控制 8

3.1 输入设备静态控制 9

3.2 输入设备动态控制 10

4 输入设备Common Feature 11

4.1 Event事件支持的接口 11

4.2 Event事件支持的委托 12

4.3 Event事件触发方式 12

4.4 API 13

4.5 PC端调试ShortKey 14

5 HandShank输入设备独有 Feature 14

5.1 独有Feature简介 14

5.2 API 14

6 Gesture26Dof输入设备独有 Feature 17

6.1 独有Feature简介 17

6.2 API 17

6.3 PC端调试ShortKey 20

7 InputSystem Feature 20

7.1 独有Feature简介 20

7.2 API 20

# 

# InputSystem简介

## 简介

InputSystem输入子系统是ShadowSDK中负责处理输入设备的子系统，此子系统负责各种输入设备的数据收集，数据处理，目标的检测，事件的派发等

InputSystem输入子系统支持多种输入设备并存及动态启用关闭，其中多种输入设备特指：

a):XR设备的6Dof数据+XR设备上的实体按键，后引述为Head输入设备

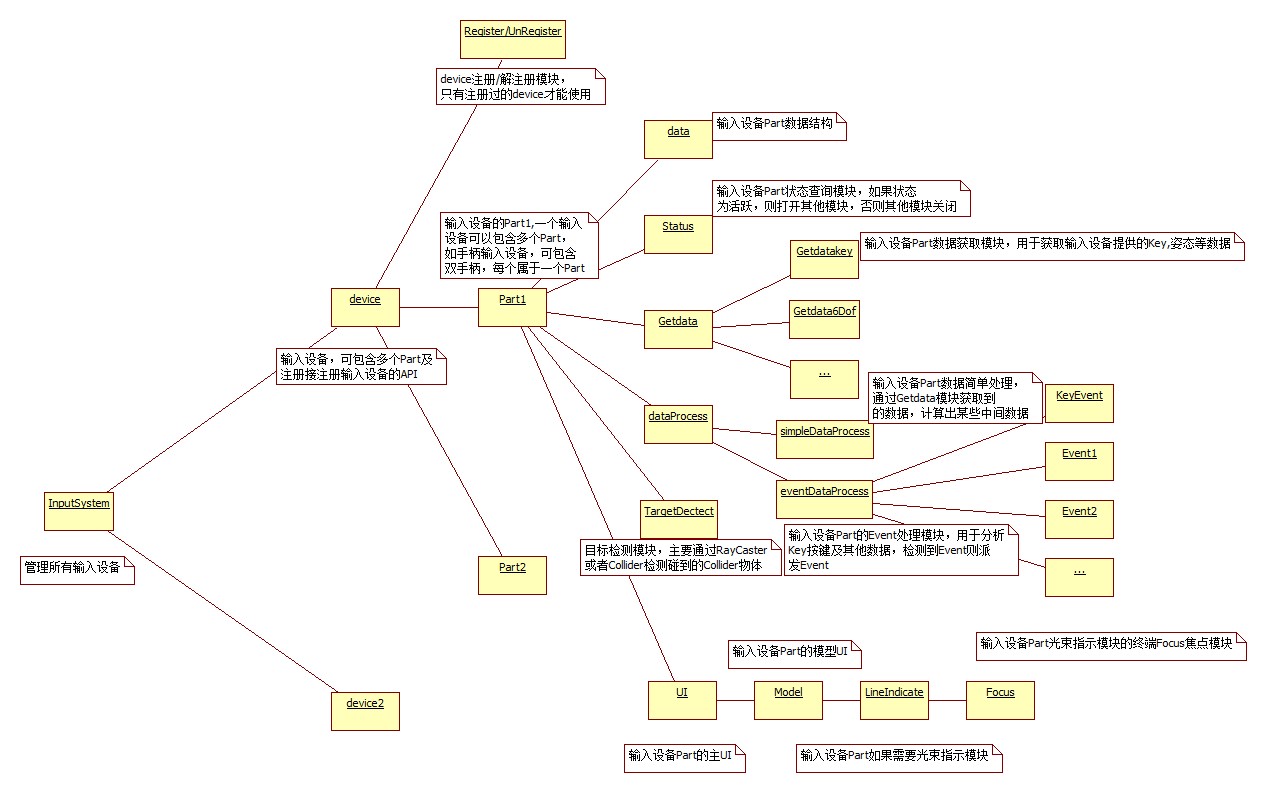
b):XR设备所支持的手柄（支持双手柄），后引述为HandShank输入设备

c):XR设备所支持的26自由度手势（支持双手），后引述为Gesture26Dof输入设备

InputSystem输入子系统意在提取各种输入设备共性，进行统一管理，对于上层处理逻辑屏蔽底层具体实现，如用输入设备进行拖拽，点击等Event操作，用户只需了解此类Event触发时回调的接口即可，具体是哪个输入设备触发则无需过多关注；

如下会详细介绍输入设备的UML及输入设备支持的特性

## UML图

****

## 概念

#### a):【InputSystem】：管理所有输入设备子系统，如注册，解注册某个输入设备等

#### b):【输入设备】：泛指一个完整的输入设备，如上文提及的Head输入设备，HandShank输入设备， Gesture26Dof输入设备等

#### c): 【输入设备Part】：具体到某个输入设备中的某部分（Part）,如HandShank输入设备中的第一个手柄，26DofGesture26Dof输入设备中的右手等

#### d): 【输入设备Register/UnRegister】：输入设备注册到InputSystem所需要的模块，争对不同输入设备具体实现

#### e): 【输入设备Part Data】：输入设备Part所需要存储数据的数据结构，管理此输入设备Part所有数据

#### f): 【输入设备Part Status】：输入设备Part状态检测模块，检测此输入设备Part是否活跃

#### g): 【输入设备Part GetData】：输入设备Part数据获取模块，用于此输入设备Part数据的获取，然后填充到输入设备Part Data中

#### h): 【输入设备Part DataProcess】：输入设备Part用于处理输入设备Part Data中数据所需要的模块，进一步会分为简单处理模块和Event处理模块（分析数据是否有Event发生并派发Event）

#### i): 【输入设备Part TargeDectect】：输入设备Part用于检测目标物体的模块，主要通过射线或者Collider检测包含Collider的目标物体，检测到目标物体后，输入设备Part DataProcess会给目标物体派发的Event

#### j): 【输入设备Part UI 】：输入设备Part所需要UI显示

## 输入设备基类简介

输入设备主要由几个类构成，如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| InputDeviceBase | [**输入设备**]抽象类，表示一个输入设备 | | | | | |
| InputDevicePartBase | [**输入设备Part**]抽象类，表示一个输入设备Part | | | | | |
| InputDataBase | [**输入设备Part**]输入的数据储存抽象类，也就是[**输入设备Part Data**] | | | | | |
| InputDataGetBase | 获取[**输入设备Part Data**]抽象类，表示获取[**输入设备Part Data**]的方法 | | | | | |
| InputDevicePartStatusBase | [**输入设备Part**]状态的判断类，用于决定输入是被是否可用 | | | | | |
| InputDataProcessBase | [**输入设备Part Data**]的处理类，分为如下两个模块 | | | | | |
| InputDataSimpleProcessBase | | [**输入设备Part Data**]简单的处理，主要是一些按键处理 | | | |
| InputDataEventProcessBase | | [**输入设备Part Data**]Event的处理，主要是检测有无Event发生 | | | |
| InputDeviceTargetDetecterBase | [**输入设备Part**]的目标检测，基本上每个[**输入设备Part**]都会有一个通过射线检测目标的需求，另外个别[**输入设备Part**]还有别的检测目标需求，如手势的每个关节 | | | | | |
| InputDeviceUIBase | [**输入设备Part**]的UI抽象类，表示[**输入设备Part**]UI,此脚本需挂在游戏物体上 | | | | | |
| Model | [**输入设备Part**]模型（后引述为[**输入设备Part模型**]）上挂载的脚本，控制模型的行为 | | | | |
| LineIndicate | | [**输入设备Part模型**]上的射线探测部分（后引述为[**射线探测**]）挂载的脚本，控制射线探测部分的行为 | | |
| Focus | [**射线探测**] 终端的[**光圈**]上挂载的脚本 | |
| EndOfPointWhenTarget | 当检测到目标时，[**光圈**]上启用的游戏对象挂在的脚本 |
| RegisterAndUnRegisterBase | [**输入设备**]注册或注销到InputSystem中的方法，不同的[**输入设备**]需要实现Register和UnRegister两个方法 | | | | | |

# 输入设备种类简介

## 输入设备中类如上介绍包括：

a):XR设备的6Dof数据+XR设备上的实体按键，后引述为Head输入设备

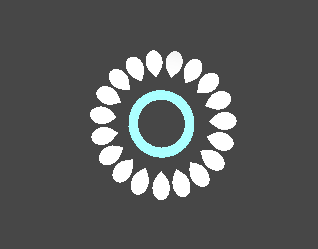
b):XR设备所支持的手柄（支持双手柄），后引述为HandShank输入设备

c):XR设备所支持的26自由度手势（支持双手势），后引述为Gesture26Dof输入设备

## Head输入设备

### Head输入设备是最简单的输入设备，由XR设备的6Dof数据+XR设备上的实体按键组成。

### 1：显示效果：

上图分别为，未检测到Collider物体，检测到Collider物体时，凝视Collider物体时的表现形式,图中光圈后续引述为**[光圈]**。

2：使用方式：

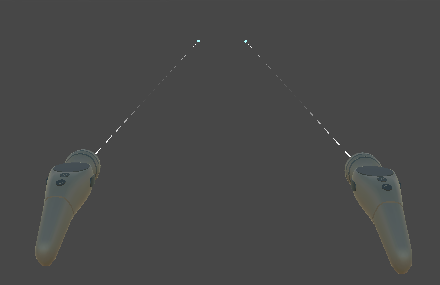
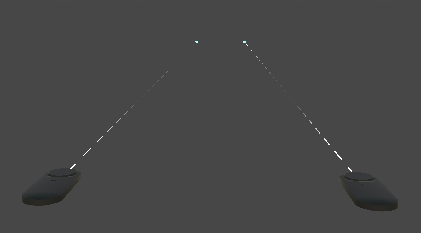
通过XR设备的移动或旋转，控制**[光圈]**的移动或旋转

通过XR设备上的实体按键（+）键发送Enter,（-）键发送Cancel键等Key Event

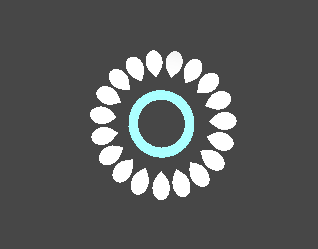
## HandShank输入设备

### HandShank输入设备是由XR设备所支持的手柄（支持双手柄）组成

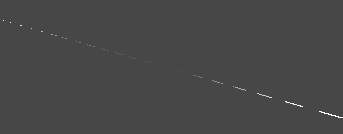
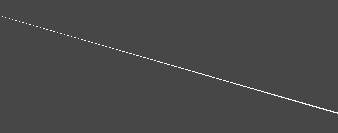
### 1：显示效果：

上图为K07手柄和K02手柄的表现形式，HandShank包含两个Part,HandShankOne/HandShankTwo

上图分别为，未检测到Collider物体，检测到Collider物体时，凝视Collider物体时**[光圈]**的表现形式。

上图分别为Trigger键未按下与按下时Ray光线的表现形式，后引述为**[Ray光线]**

2：使用方式：

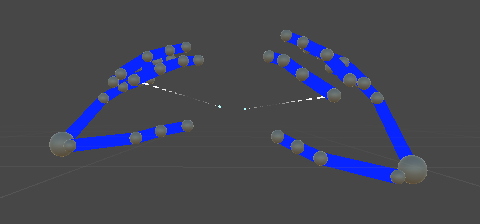
通过XR设备所支持的手柄旋转，控制**[光圈]及[Ray光线]**的旋转

通过XR设备所支持的手柄上的实体按键发送Key Event

## Gesture26Dof输入设备

### Gesture26Dof输入设备是由所支持的26自由度手势（支持双手势）组成

### 1：显示效果：

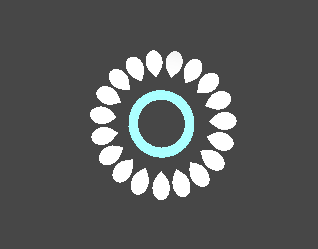


上图为双手势的表现形式，其中包括Gesture26Dof输入设备的两个Part，每个Part都包含如下内容：

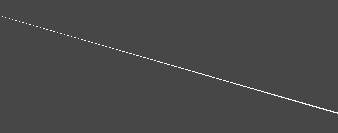
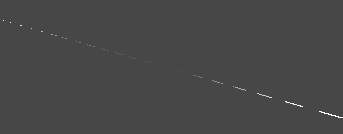
手势轮廓：如上蓝色线条，表示每个finger，后引述为**[手势轮廓]**,可以通过API控制显示与否；

手势关节：如上圆球体，表示每个finger上的每个joint，后引述为**[手势关节]**，可以通过API控制显示与否；

手势射线：也就是手势**[Ray光线]**，可以通过API控制显示与否；

上图分别为，未检测到Collider物体，检测到Collider物体时，凝视Collider物体时**[光圈]**的表现形式。



上图分别为手势抓取或捏住时**[Ray光线]**的表现形式

2：使用方式：

通过XR设备所支持的26自由度手势移动和XR设备的旋转，控制**[光圈]及[Ray光线]**的移动及旋转

通过XR设备所支持的26自由度手势进行抓取或者捏取动作，触发Key Event

# 输入设备控制

## 输入设备控制指Enable/Disable某个/些输入设备及其所属Part，分为静态控制和动态控制，静态

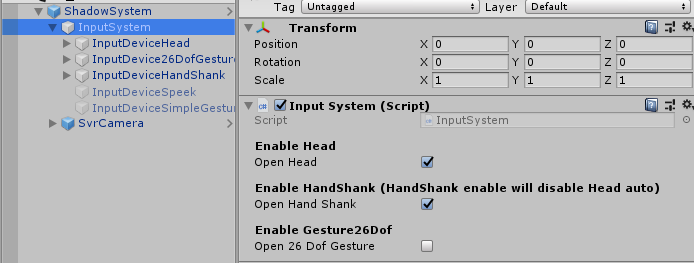
## 控制指程序运行前就设定，动态控制指程序运行中设定

## 输入设备静态控制

### 静态控制Enable/Disable某个/些输入设备

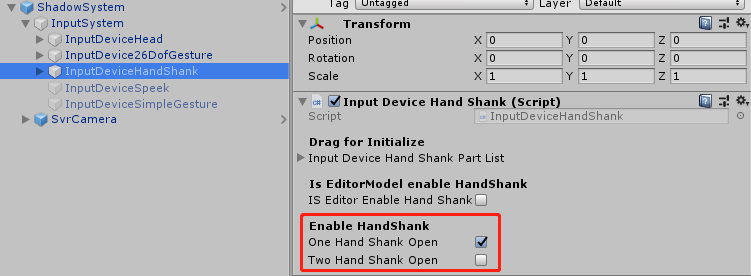
静态控制只需设定InputSystem的一些属性即可，如下图：

找到Scene中的ShadowSystem游戏对象下的InputSystem，再Inspector视图中将需要启用的输入设备勾选即可



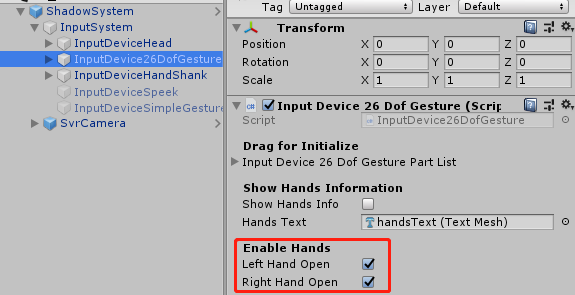
### 静态控制Enable/Disable某个/些输入设备所属Part

#### HandShank输入设备所属的HandShankOne与HandShankTwo两个Part可以通过如下属性控制



备注：上图中IsEditorEnableHandShank属性用于在编辑器模式下关闭或打开手柄，此属性不影响在非编辑器平台上的效果

Gesture26Dof输入设备所属的HandLeft和HandRight两个Part可以通过如下属性控制



## 输入设备动态控制

### 动态控制Enable/Disable某个/些输入设备

通过如下注册/解注册输入设备API可以实现输入设备的动态控制:

InputSystem.Instant.RegisterInputDevice(InputDeviceType);

InputSystem.Instant.UnRegisterInputDevice(InputDeviceType);

备注：当输入设备数据有效时自动启用输入设备

备注：此接口宜在Start方法中使用，Awake时InputSystem.Instant正在进行初始化

输入设备API：

其中 InputDeviceType 包含如下输入设备

public enum InputDeviceType {

UnKnow,

Head,

HandShank,

Gesture26DofHand,

GestureSimple,

Speek,

Voice,

## }

### 动态控制Enable/Disable某个/些输入设备所属Part

通过设定如下属性可动态控制输入设备所属Part的启用关闭

HandShank所属Part:

InputSystem.Instant.HandShank.OneHandShankOpen

InputSystem.Instant.HandShank.TwoHandShankOpen

Gesture26Dof所属Part:

InputSystem.Instant.Gesture26Dof.LeftHandOpen

InputSystem.Instant.Gesture26Dof.RightHandOpen

# 输入设备Common Feature

Common Feature顾名思义就是所有输入设备都具有的特性

## Event事件支持的接口

如下为各个输入设备Event事件支持的接口：

可以通过继承抽象类PointerHandlers或者直接实现接口达到监听Event的目的

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **输入设备** | | | **Event事件** | **接口** |  |
| HandShank输入设备 | Gesture26Dof输入设备 | Head输入设备 | OnSCPointerEnter | ISCPointerEnterHandler |  |
| OnSCPointerDown | ISCPointerDownHandler |
| OnSCPointerClick | ISCPointerClickHandler |
| OnSCPointerDrag | ISCPointerDragHandler |
| OnSCPointerUp | ISCPointerUpHandler |
| OnSCPointerExit | ISCPointerExitHandler |
| OnPointerEnter | IPointerEnterHandler |
| OnPointerDown | IPointerDownHandler |
| OnPointerClick | IPointerClickHandler |
| OnPointerDrag | IDragHandler |
| OnPointerUp | IPointerUpHandler |
| OnPointerExit | IPointerExitHandler |
|  | | OnBackKeyDown | IBackKeyDownHandler |  |
| OnBackKeyUp | IBackKeyUpHandler |
| OnTriggerKeyDown | ITriggerKeyDownHandler |
| OnTriggerKeyUp | ITriggerKeyUpHandler |
| OnFunctionKeyDown | IFunctionKeyDownHandler |
| OnFunctionKeyUp | IFunctionKeyUpHandler |
| OnTouchKeyDown | ITouchKeyDownHandler |
| OnTouchKeyUp | ITouchKeyUpHandler |
| OnVolumeDownKeyDown | IVolumeDownKeyDownHandler |
| OnVolumeDownKeyUp | IVolumeDownKeyUpHandler |  |
| OnVolumeUpKeyDown | IVolumeUpKeyDownHandler |
| OnVolumeUpKeyUp | IVolumeUpKeyUpHandler |
|  |  |

## Event事件支持的委托

### 如下为各个输入设备Event事件支持的委托：

### 可以通过继承抽象类PointerDelegate或者直接添加委托达到监听Event的目的

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **输入设备** | | | **Event事件** |  |
| HandShank输入设备 | Gesture26Dof输入设备 | Head输入设备 | InputDeviceEventKeyBase.partAnyKeyDownDelegate |  |
| InputDeviceEventKeyBase.partAnyKeyUpDelegate |
| InputDeviceEventKeyBase.partAnyKeyLongDelegate |
| InputDeviceEventKeyBase.partEnterKeyDownDelegate |
| InputDeviceEventKeyBase.partEnterKeyUpDelegate |
| InputDeviceEventKeyBase.partEnterKeyDragDelegate |
|  | | InputDevice26DofGestureEventBase.eventDelegate |  |

## Event事件触发方式

### Event事件分为Key Event及其他Event，如手势中的TurnDown等Event

### 如下主要介绍Key Event的触发方式:

### 1：Head 输入设备的Key Event：

### 通过XR设备上的实体按键（+）键发送Enter,（-）键发送Cancel键等Key Event

### 2：HandShank输入设备的Key Event:

### XR设备所支持的手柄上的实体按键发送Key Event

### 3：Gesture26Dof输入设备的Key Event:

### XR设备所支持的26自由度手势进行抓取或者捏取动作，触发Key Event

## API

1):获取某个输入设备某个Part的数据API：

以HandShank的partHandShankOne为例：

if(InputSystem.Instant.HandShank != null) {

foreach(var part in InputSystem.Instant.HandShank.inputDeviceHandShankPartList) {

if(part.PartType == InputDevicePartType. HandShankOne) {

if(part.inputDataHandShank.isVaild) {

transform.rotation = part.inputDataHandShank.rotation;

}

}

}

}

可以获取的Part的数据有：

1：part当前数据是否有效，当无效时，此Part的UI将不显示：

part.inputDataHandShank.isVaild

2：part当前点击后的数据：

part.inputDataHandShank. pointerEventData

3：Part当前所有按键实时状态字典：

part.inputDataHandShank. inputKeyDic

3：Part当前所有按键按下状态字典：

part.inputDataHandShank. inputKeyPressDic

4：Part当前某个按键Down/Up状态：

part.inputDataHandShank. GetKeyDown(InputKeyCode inputKeyCode)

part.inputDataHandShank. GetKeyUp(InputKeyCode inputKeyCode)

## PC端调试ShortKey

### 1：视角旋转

快捷键1：鼠标右键长按+移动鼠标

快捷键2：鼠标左键长按+移动鼠标

2：视角移动

WSAD分别控制视角前进后退左右移动

3：发送确认键

鼠标左键点击

4：发送拖拽

鼠标左键长按+移动鼠标

# HandShank输入设备独有 Feature

## 独有Feature简介

### a):支持双手柄

### 双手柄特指手柄1号/手柄2号，而不是左手柄/右手柄，首先连接的手柄会自动识别为手柄1号，第二个连接的手柄自动识别为手柄2号，提供API动态关闭手柄1号或者手柄2号；

### b):手柄如果连接上则自动使得Head输入设备数据无效，也就是说手柄连接上，Head输入设备自动隐藏；

### c):双手柄可与Gesture26Dof输入设备共存；

## API

### a):启用/关闭HandShank输入设备API

InputSystem.Instant.RegisterInputDevice(InputDeviceType.HandShank);

InputSystem.Instant.UnRegisterInputDevice(InputDeviceType.HandShank);

### 备注：此接口宜在Start方法中使用，Awake时InputSystem.Instant正在进行初始化

### 备注：启用后并不是就可以使用，需要手柄连接设备，

### 备注：启用HandShank输入设备后Head输入设备自动隐藏，关闭HandShank输入设备后Head输入设备自动开启

### b):获取HandShank输入设备API:

InputSystem.Instant.HandShank;

### 备注：当此为Null表示HandShank未启用或者启用中（InputSystem.Instant.RegisterInputDevice并不能立即完成）

### c):启用/关闭HandShank的某个Part(手柄1号并/或手柄2号)API

InputSystem.Instant.HandShank.OneHandShankOpen;

InputSystem.Instant.HandShank.TwoHandShankOpen;

### 备注：true表示开启此Part,false表示关闭此Part,设置后的下一帧生效；

### 备注：需要判断InputSystem.Instant.HandShank是否为Null

### D):获取HandShank所有支持的Part的API

InputSystem.Instant.HandShank.inputDeviceHandShankPartList;

### 备注：可以遍历此list,然后更加partType属性获得此Part的类型，HandShank暂时支持2Part类型：HandShankOne, HandShankTwo；

### e):获取HandShank某个Part的数据API：

### 以HandShank的partHandShankOne为例：

if(InputSystem.Instant.HandShank != null) {

foreach(var part in InputSystem.Instant.HandShank.inputDeviceHandShankPartList) {

if(part.PartType == InputDevicePartType. HandShankOne) {

if(part.inputDataHandShank.isVaild) {

transform.rotation = part.inputDataHandShank.rotation;

}

}

}

}

### 可以获取的Part的数据有：

### 1：part当前数据是否有效，当无效时，此Part的UI将不显示：

part.inputDataHandShank.isVaild

### 2：part当前点击后的数据：

part.inputDataHandShank. pointerEventData

### 3：Part当前所有按键实时状态字典：

part.inputDataHandShank. inputKeyDic

### 3：Part当前所有按键按下状态字典：

part.inputDataHandShank. inputKeyPressDic

### 4：Part当前某个按键Down/Up状态：

part.inputDataHandShank. GetKeyDown(InputKeyCode inputKeyCode)

part.inputDataHandShank. GetKeyUp(InputKeyCode inputKeyCode)

### 5：Part对应的手柄是否连接：

InputDataHandShank. HSData.HS0Connect

InputDataHandShank. HSData.HS1Connect

### 6：Part类型，分为K07/K02：

part.inputDataHandShank.handShankType

### 7：Part连接手柄的标识符，分为“K07””K02”：

part.inputDataHandShank.blueToothName

### 8：Part的TP坐标数据：

part.inputDataHandShank.tpPosition

### 9：Part的Tp接触状态：

part.inputDataHandShank.isTpTouch

### 10：Part的旋转数据：

part.inputDataHandShank.rotation

# Gesture26Dof输入设备独有 Feature

## 独有Feature简介

### a):支持双手势

### 双手势特指是左手势/右手势，提供API动态关闭左手势或者右手势；

### b): 任意手势如果识别到则自动使得Head输入设备数据无效，也就是说左或/并右手势识别到，Head输入设备自动隐藏；左并右手势丢失，则Head输入设备自动开启

### c):双手柄可与Gesture26Dof输入设备共存；

### d):手势支持射线检测物体（后引述为Ray方式）或者触摸检测物体（后引述为Touch方式）两种方式，若两种同时检测到物体，则优先使用Touch方式检测的物体；提供API关闭/开启检测方式；

## API

### a):启用/关闭Gesture26Dof输入设备API

InputSystem.Instant.RegisterInputDevice(InputDeviceType. Gesture26DofHand);

InputSystem.Instant.UnRegisterInputDevice(InputDeviceType. Gesture26DofHand);

### 备注：此接口宜在Start方法中使用，Awake时InputSystem.Instant正在进行初始化

### 备注：启用后将手放于RGBcamera视角范围则可以看到手势出现

### 备注：手势需要开启Camera/外存访问权限，可自行到设置->权限中设置

### 备注：启用Gesture26Dof输入设备后，只要识别到任意一只手，Head输入设备自动隐藏，双手都丢失后后Head输入设备自动开启

### b):获取Gesture26Dof输入设备API:

InputSystem.Instant. Gesture26Dof;

### 备注：当此为Null表示Gesture26Dof未启用或者启用中（InputSystem.Instant.RegisterInputDevice并不能立即完成）

### c):启用/关闭Gesture26Dof的某个Part(左手势或/并右手势)API

InputSystem.Instant.HandShank. LeftHandOpen;

InputSystem.Instant.HandShank. RightHandOpen;

### 备注：true表示开启此Part,false表示关闭此Part,设置后的下一帧生效；

### 备注：需要判断InputSystem.Instant.Gesture26Dof是否为Null

### D):获取Gesture26Dof所有支持的Part的API

InputSystem.Instant.HandShank. InputDevice26DofGesturePartList;

### 备注：可以遍历此list,然后更加partType属性获得此Part的类型，Gesture26Dof暂时支持2Part类型：Gesture26DofHandLeft, Gesture26DofHandRight；

### E):获取Gesture26Dof某个Part的数据API：

### 以Gesture26Dof的Gesture26DofHandLeft为例：

if(InputSystem.Instant. Gesture26Dof!= null) {

foreach(var part in InputSystem.Instant. Gesture26Dof. InputDevice26DofGesturePartList) {

if(part.PartType == InputDevicePartType. Gesture26DofHandLeft) {

if(part. inputData26DofGesture.isVaild) {

transform. position = part. inputData26DofGesture. handInfo. position;

}

}

}

}

### 可以获取的Part的数据有：

### 1：part当前数据是否有效，当无效时，此Part的UI将不显示：

part.inputDataHandShank.isVaild

### 2：part当前点击后的数据：

part.inputDataHandShank. pointerEventData

### 3：Part当前所有按键实时状态字典：

part.inputDataHandShank. inputKeyDic

### 3：Part当前所有按键按下状态字典：

part.inputDataHandShank. inputKeyPressDic

### 4：Part当前某个按键Down/Up状态：

part.inputDataHandShank. GetKeyDown(InputKeyCode inputKeyCode)

part.inputDataHandShank. GetKeyUp(InputKeyCode inputKeyCode)

### 5：part当前是否丢失：

part. inputData26DofGesture. handInfo. isLost

### 备注：true表示丢失，false表示未丢失或丢失中

### 6：Part当前丢失进度：

part. inputData26DofGesture. handInfo. lostPercent

### 备注：取值范围0-1，浮点型，1表示完全丢失

### 7：Part当前丢失设置的时间：

part. inputData26DofGesture. handInfo. lostTimer

### 备注：范围秒，浮点型

### 8：Part的手指数据：

part. inputData26DofGesture. handInfo. Finger

### 备注：例如获取大拇指的第一个关节的检测到的目标：

part. inputData26DofGesture. handInfo. Finger[(int) FINGER. thumb]. Joint[(int) JOINT. One].target

## PC端调试ShortKey

### 1：发送抓取

按键1控制左手抓取，按键2控制右手抓取

2：手势丢失

按键O控制左手丢失，按键P控制右手丢失

3：手势位置

按键IKJL分别控制右手势前进后退左右移动

# InputSystem Feature

## Feature简介

### InputSystem管理所有的输入设备，包括注册/注销。动态启用某个输入设备

## API

### a): InputSystem系统的单例引用

### public static InputSystem Instant;

### b): 获取HandShank输入设备

### public InputDeviceHandShank.InputDeviceHandShank HandShank

### c): 获取Head输入设备

### public InputDeviceHandShank.InputDeviceHandShank HandShank

### d): 获取Gesture26Dof输入设备

### public InputDevice26DofGesture.InputDevice26DofGesture Gesture26Dof

### d): 注册输入设备

### public Coroutine RegisterInputDevice(InputDeviceType inputDeviceType)

### d): 解注册输入设备

### public Coroutine UnRegisterInputDevice(InputDeviceType inputDeviceType)