Azure Kinect Examples for Unity

Table of contents

Introduction to 'Azure Kinect Examples' v1.16 On-line Documentation

This is the on-line documentation of "Azure Kinect Examples for Unity" (or K4A-asset for short). It describes the demo scenes in the package, the general-purpose components in KinectScripts, as well as the demo-specific components, used by the demo scenes. The component documentation includes description of the respective component and short descriptions of its public properties, as well.

How to Run 'Azure-Kinect Examples for Unity'

Despite its name, 'Azure-Kinect Examples for Unity' can work with several depth sensors – Azure-Kinect, RealSense and Kinect-v2. The installation depends on what sensor you have at your disposal.

1. (Azure Kinect) Download and install Azure-Kinect Sensor SDK, as described in the 'Azure-Kinect SDKs'-section below.
2. (Azure Kinect) Download and install Azure-Kinect Body Tracking SDK, as described in the 'Azure-Kinect SDKs'-section below.
3. (Kinect-v2) Download and install Kinect SDK 2.0, as described in the 'Kinect-v2 SDK'-section below.
4. (RealSense) Download and install RealSense SDK 2.0, as described in the 'RealSense SDK'-section below.
5. Import this package into new Unity project.
6. Open 'File / Build settings' and switch to 'PC, Mac & Linux Standalone', target platform: 'Windows'.
7. Make sure that Direct3D11 is the first option in the 'Auto Graphics API for Windows'-list setting, in 'Player Settings / Other Settings / Rendering'.
8. Open and run a demo scene of your choice from a subfolder of the 'AzureKinectExamples/KinectDemos'-folder. Short descriptions of the available demo scenes will be published soon.

Installation of Azure-Kinect SDKs

1. Download the latest version of 'Azure Kinect Sensor SDK'. The download page is below.
2. Run the installer. The installation of 'Azure Kinect Sensor SDK' is straightforward.
3. Connect the Azure Kinect sensor.
4. Run 'Azure Kinect Viewer' to check, if the sensor and SDK work as expected.
5. Download the latest version of 'Azure Kinect Body Tracking SDK'. The download page is below.
6. Follow the instructions, on how to install the body tracking SDK. See the link below.
7. Run 'Azure Kinect Body Tracking Viewer' and move in front of the sensor to check, if the body tracking SDK works as expected.

* The latest Azure Kinect Sensor SDK (v1.4.1) can be found here.
* The latest Azure Kinect Body Tracking SDK (v1.1.0) can be found here.
* Older releases of Azure Kinect Body Tracking SDK can be found here.
* Instructions how to install the body tracking SDK can be found here.

Installation of Kinect-v2 SDK

1. Download the Kinect for Windows SDK 2.0. Here is the download page.
2. Run the installer. Installation of Kinect SDK 2.0 is straightforward.
3. Connect the Kinect-v2 sensor. The needed drivers will be installed automatically.
4. Run 'SDK Browser 2.0' and select 'Kinect Configuration Verifier' to check, if the sensor and SDK work as expected.

Installation of RealSense SDK

1. Download the latest version of RealSense SDK 2.0. Here is the download page.
2. Run the installer. Installation of RealSense SDK 2.0 is straightforward.
3. Connect the RealSense sensor.
4. Run 'Intel RealSense Viewer' to check, if the sensor and SDK work as expected.

Short Descriptions of the Demo Scenes

Here are short descriptions of the available demo scenes, along with the major components they utilize and demonstrate.

Scene:	***Description:***
AvatarDemo / KinectAvatarsDemo1	The scene shows two avatars controlled by the user, from third person perspective. The scene utilizes the KinectManager-component to manage the sensor and data, AvatarController-components to control each of the two avatars, as well as SimpleGestureListener-component to demonstrate the gesture detection process.
AvatarDemo / KinectAvatarsDemo2	It demonstrates avatar control from first-person perspective. The scene utilizes the KinectManager-component to manage the sensor and data and the AvatarController-component to control the 1st person avatar.
AvatarDemo / KinectAvatarsDemo3	This scene utilizes the AvatarControllerClassic-component, to control the avatar's upper body, as well as offset node to make the avatar motion relative to a game object. The scene utilizes the KinectManager-component to manage the sensor and data, and AvatarControllerClassic-component to control the upper body of the avatar.
AvatarDemo / KinectAvatarsDemo4	This demo shows how to instantiate and remove avatars in the scene, to match the users in front of the camera. It utilizes the KinectManager-component to manage the sensor and data, UserAvatarMatcher-component to instantiate and remove the avatars in the scene as the users come and go, as well as AvatarController-components to control each of the instantiated avatars.
BackgroundRemovalDemo / BackgroundRemovalDemo1	This background-removal demo shows how to display the user silhouettes on a virtual background (the so called green screen). It utilizes the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render the user silhouettes, and PortraitBackground-component to manage different screen resolutions.
BackgroundRemovalDemo / BackgroundRemovalDemo2	This demo shows how to display part of the real environment on a virtual background. It utilizes the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render the real environment in the scene, BackgroundRemovalByDist-component to filter a part of the real environment and PortraitBackground-component to manage different screen resolutions.
BackgroundRemovalDemo / BackgroundRemovalDemo3	This demo scene shows how to display the user silhouettes on a virtual background by utilizing the detected body indices. It utilizes the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render the user silhouettes, BackgroundRemovalByBodyIndex-component to filter the user silhouettes, according to the detected body indices, and PortraitBackground-component to manage different screen resolutions.
BackgroundRemovalDemo / BackgroundRemovalDemo4	This scene demonstrates how to display part of the virtual scene environment within the user's silhouette. It utilizes the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render user silhouettes, and PortraitBackground-component to manage different screen resolutions.

BackgroundRemovalDemo / BackgroundRemovalDemo5	This demo scene shows how to display the user silhouettes in 3D virtual environment, according to user's distance to the sensor. It utilizes the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render the user silhouettes, ForegroundToRenderer-component to render the user's silhouette on a game object, UserImageMover-component to move the user's image object according to user's distance to the sensor, ColorImageJointOverlayer-component to overlay user's joint with a virtual object, and PortraitBackground-component to manage different screen resolutions.
BlobDetectionDemo / BlobDetectionDemo	The blob-detection demo shows how to detect blobs (compact areas) of pixels in the raw depth image, within the min/max distance configured for the sensor. It utilizes the KinectManager-component to manage the sensor and data, BlobDetector-component to detect the blobs in the raw depth image, coming from the sensor, and BackgroundDepthImage-component to display the depth camera image on the scene background.
BlobDetectionDemo / DepthIrFilterDemo	The IR depth-filter demo shows how to filter the IR image with the raw depth image data, within the min/max distance configured for the sensor. It utilizes the KinectManager-component to manage the sensor and data, and DepthIrFilterImage-component to filter the IR image with the raw depth data, and display it on the scene background.
ColliderDemo / ColorColliderDemo	It demonstrates how to trigger ‘virtual touch’ between the user’s hands and virtual scene objects. The scene utilizes the KinectManager-component to manage the sensor and data, HandColorOverlayer-component to move the overlaying hand-objects with colliders, JumpTrigger-component to detect virtual object collisions, and BackgroundColorImage-component to display the color camera image on the scene background.
ColliderDemo / DepthColliderDemo2D	It shows how the user’s silhouette can interact with virtual objects in 2D-scene. This scene utilizes the KinectManager-component to manage the sensor and data, DepthSpriteViewer-component to display the user’s silhouette and create the overlaying skeleton colliders, and EggSpawner-component to spawn the virtual objects (spheres) in the scene.
ColliderDemo / DepthColliderDemo3D	It shows how the user’s silhouette can interact with virtual objects in 3D-scene. The scene utilizes the KinectManager-component to manage the sensor and data, DepthImageViewer-component to display the user’s silhouette and create the overlaying skeleton colliders, and EggSpawner-component to spawn the virtual objects (eggs) in the scene.
ColliderDemo / SkeletonColliderDemo	This scene shows how the user’s skeleton can interact with virtual objects in the scene. It utilizes the KinectManager-component to manage the sensor and data, SkeletonCollider-component to display the user’s skeleton and create bone colliders, and BallSpawner-component to spawn virtual objects in the scene.
FittingRoomDemo / KinectFittingRoom1	This is the main dressing room demo-scene. It demonstrates the use of calibration pose, model overlay and blending between virtual and physical objects. It utilizes the KinectManager-component to manage the sensor and data, FollowSensorTransform-component to keep the camera pose in sync with the sensor, BackgroundColorImage-component to display the color camera feed on BackgroundImage2, CategorySelector-component to change model categories as needed, ModelSelector-component to manage model menu population and model selection, PhotoShooter-component to manage making photos, and UserBodyBlender-component to blend the virtual with physical objects.
FittingRoomDemo / KinectFittingRoom2	This is the second dressing room demo-scene. It shows how to overlay the user's body with an arbitrary humanoid model (like Ironman, Cinderella, Ninja turtle, etc.). The scene utilizes the KinectManager-component to manage the sensor and data, FollowSensorTransform-component to keep the camera pose in sync with the sensor, AvatarController-component to control the virtual model, AvatarScaler-component to scale the model according to the user's dimensions, BackgroundColorImage-component to display the color camera feed on BackgroundImage2, and optionally UserBodyBlender-component to blend the virtual with physical objects.

GestureDemo / KinectGesturesDemo1	This scene demonstrates the detection of discrete gestures (swipe-left, swipe-right & swipe up in this demo), used to control a presentation cube. The scene utilizes the KinectManager-component to manage the sensor and data, CubeGestureListener-component to listen for swipe-gestures, and CubePresentationScript-component to control the presentation cube in the scene.
GestureDemo / KinectGesturesDemo2	It demonstrates the detection of continuous gestures (wheel, zoom-out & zoom-in in this demo), used to rotate and zoom a 3D model. The scene utilizes the KinectManager-component to manage the sensor and data, ModelGestureListener-component to set up and listen for wheel and zoom gestures, and ModelPresentationScript-component to control the 3D model in the scene.
GreenScreenDemo / GreenScreenDemo1	The scene shows how to utilize a green screen for background segmentation. The scene uses the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render the real environment in the scene, BackgroundRemovalByGreenScreen-component to filter a part of the real environment, according to its similarity or difference to the color of the green screen, and PortraitBackground-component to manage different screen resolutions.
GreenScreenDemo / GreenScreenDemo2	It shows how to utilize a green screen for background segmentation and volumetric rendering. The scene uses the KinectManager-component to manage the sensor and data, BackgroundRemovalManager-component to render the real environment in the scene, BackgroundRemovalByGreenScreen-component to filter a part of the real environment, according to its similarity or difference to the color of the green screen, ForegroundBlendRenderer-component to provide volumetric rendering and lighting of the filtered environment in the scene, and PortraitBackground-component to manage different screen resolutions.
InteractionDemo / KinectInteractionDemo1	This scene demonstrates the hand cursor control on screen and hand interactions, in means of grips and releases. The scene utilizes the KinectManager-component to manage the sensor and data, InteractionManager-component to manage the hand cursor and user interactions, GrabDropScript-component to interact with virtual objects in the scene via hand grips & releases, and InteractionInputModule to convey the user interactions with the UI to the Unity event system.
InteractionDemo / KinectInteractionDemo2	It shows how to use hand interactions to grab and turn a virtual object on screen in all directions. The scene utilizes the KinectManager-component to manage the sensor and data, InteractionManager-component to manage the hand cursor and user interactions, and GrabRotateScript-component to interact with the virtual object in the scene.
MocapAnimatorDemo / MocapAnimatorDemo	The Mocap-animator scene provides a simple tool to capture the user's motion as Unity animation. The scene utilizes the KinectManager-component to manage the sensor and data, MocapRecorder-component to capture and record the user's motions as Unity animation asset, MocapPlayer-component to play the recorded animation on a humanoid model and AvatarController-component to provide the motion data.
MultiCameraSetup / MultiCameraSetup	The Multi-Camera-Setup scene provides a tool to estimate the sensor positions and rotations in a multi-camera setup. It utilizes the KinectManager-component to manage the sensor and data, MultiCameraSetup-component to estimate the positions and rotations of the sensors, and UserMeshRendererGpu-component to render the user in the scene, as perceived by the sensors.
MultiSceneDemo / Scene0-StartupScene, Scene1-AvatarsDemo & Scene2-GesturesDemo	This set of scenes shows how to use the KinectManager and other Kinect-related components across multiple scenes. It utilizes the KinectManager-component to manage the sensor and data, LoadFirstLevel-component in the startup scene to load the first real scene, LoadLevelWithDelay-component in the real scenes to cycle between scenes, and RefreshGestureListeners-component to refresh the list of gesture listeners for the current scene. The other Kinect-related components like AvatarController, gesture listeners, etc. are utilized in the real scenes (1 & 2), but they are related to the respective scene specifics, and not to using a single KinectManager across multiple scenes.
NetworkDemo / KinectNetServer	This scene acts as network server for the streams of the connected sensor. It

	utilizes the KinectManager-component to manage the sensor and data, KinectNetServer-component to listen for clients and send sensor data over the network, and UserAvatarMatcher-component to instantiate and remove the avatars in the scene as the users come and go.
NetworkDemo / NetClientDemo1	This scene demonstrates how to use the network-client sensor (in combination with KinectNetServer), to remotely control the avatars in the scene. It utilizes the KinectManager-component to manage the sensor and data, NetClientInterface-component to act as "network" sensor by receiving the remote sensor's data from the server over the network, and UserAvatarMatcher-component to instantiate and remove the avatars in the scene as the users come and go.
OverlayDemo / KinectOverlayDemo1	This is the most basic joint-overlay demo, showing how a virtual object can overlay the user's joint (right shoulder) on screen. The scene utilizes the KinectManager-component to manage the sensor and data, JointOverlayer-component to overlay the user's body joint with the given virtual object, and BackgroundColorImage-component to display the color camera image on the scene background.
OverlayDemo / KinectOverlayDemo2	This is a skeleton overlay demo, with green balls overlaying the body joints, and lines between them to represent the bones. The scene utilizes the KinectManager-component to manage the sensor and data, SkeletonOverlayer-component to overlay the user's body joints with the virtual objects and lines, and BackgroundColorImage-component to display the color camera image on the scene background.
OverlayDemo / KinectOverlayDemo3	This is a hand overlay demo, painting lines on screen when the user's right hand is closed. The scene utilizes the KinectManager-component to manage the sensor and data, InteractionManager-component to manage the hand interactions, HandOverlayer-component to overlay the user's hand with the given cursors and to capture the hand events, LinePainter-component to draw the lines when user's hand is closed, and BackgroundColorImage-component to display the color camera image on the scene background.
OverlayDemo / KinectPhotoBooth	This is a photo booth demo, overlaying specific user's body joints with 2d images. The scene utilizes the KinectManager-component to manage the sensor and data, InteractionManager-component to manage the hand interactions, JointOverlayer-component to overlay the user's body joint with 2d images, PhotoBoothController-component to detect user's swipe gestures and update the overlay images accordingly, and BackgroundColorImage-component to display the color camera image on the scene background.
PointCloudDemo / SceneMeshDemo	This demo shows how to integrate part of the real environment as point-cloud into the Unity scene at runtime. It utilizes the KinectManager-component to manage the sensor and data, SceneMeshRendererGpu-component to render the real environment into the scene, and FollowSensorTransform-component to move the mesh center according to sensor's position and rotation.
PointCloudDemo / UserMeshDemo	This demo shows how to integrate the real user as point-cloud into the Unity scene at runtime. It utilizes the KinectManager-component to manage the sensor and data, UserMeshRendererGpu-component to render the user into the scene, and FollowSensorTransform-component to move the mesh center according to sensor's position and rotation.
PointCloudDemo / VfxPointCloudDemo	This scene demonstrates how to combine the spatial and color data provided by the sensor with visual effect graphs to create point cloud of the environment, with or without visual effects. It utilizes the KinectManager-component to manage the sensor and data, as well as the sensor interface settings to create the attribute textures needed by the VFX graph. Please see 'How to run VfxPointCloudDemo-scene'-section below.
PoseDetectionDemo / PoseDetectionDemo1	This is simple, static pose-detection demo. It calculates the differences between the bone orientations of a static pose-model and the user, and estimates the pose matching factor. The scene utilizes the KinectManager-component to manage the sensor and data, StaticPoseDetector-component to estimate the pose-matching factor, PoseModelHelper-component to provide the model bone orientations, and AvatarController-component to control the user's avatar.
PoseDetectionDemo / PoseDetectionDemo2	In comparison to the previous demo, this demo scene estimates the pose matching factor between the user and an animated pose model. The moving-pose detection

	may be divided in steps, to provide better feedback to the user. The scene utilizes the KinectManager-component to manage the sensor and data, DynamicPoseDetector-component to estimate the pose-matching factor between the user and the animated model, MovingPoseManager-component to estimate the user performance in the given sequence of animated steps, PoseModelHelper & PoseModelHelperClassic-components to estimate the user's and model's bone orientations, and AvatarController-component to control the user's avatar.
RecorderDemo / BodyDataRecorderDemo	This is simple body-data recorder and player. The recorded body-data file may be played later on the same or other machine. The scene utilizes the KinectManager-component to manage the sensor and data, BodyDataRecorderPlayer-component to record or play the body data to or from a file, BodyDataPlayerController-component to process the key presses and start/stop the recorder or player, and UserAvatarMatcher-components to instantiate and remove avatars in the scene as needed.
RecorderDemo / PlayerDetectorDemo	This scene demonstrates how to play a body-data recording, when no user is detected for a while. The scene utilizes the KinectManager-component to manage the sensor and data, BodyDataRecorderPlayer-component to play the recorded body data from a file, PlayerDetectorController-component to check for users and start/stop the body-data player accordingly, and UserAvatarMatcher-components to instantiate and remove avatars in the scene as needed.
VariousDemos / HeightEstimatorDemo	The height estimator demo shows how to estimate user's height and some body sizes, based on the raw depth image data only. It utilizes the KinectManager-component to manage the sensor and data, BodySlicer-component to estimate user's height and other body sizes from the raw depth data, HeightEstimator-component to display the user's body info on screen, and BackgroundDepthImage-component to display the depth camera image on the scene background.
VariousDemos / HolographicViewerDemo	The holo-viewer demo scene shows how to render simple holographic scene on screen, based on the viewer's position. It utilizes the KinectManager-component to manage the sensor and data, and SimpleHolographicCamera-component to emulate 3d holographic display, depending on the viewer's position in front of the display.

How to run the VfxPointCloudDemo-scene

There are some special preparations needed for the VfxPointCloudDemo-scene. Please do as follows:

1. Run the Unity package manager (menu Window / Package Manager). Select to view ‘All packages’ and enable the ‘Show preview packages’-option.
2. Install the ‘High Definition RP’-package.
3. Install the ‘Visual Effect Graph’-package, as well.
4. Open the project settings (menu Edit / Project Settings) and select ‘Graphics’. Then drag the HDRP-asset from the KinectDemos/PointCloudDemo/HDRP-folder of the project to the ‘Scriptable Render Pipeline Settings’ in the Graphics-section.
5. Select the ‘Player’-section on the left. Change the color space from ‘Gamma’ to ‘Linear’.
6. Open and run the VfxPointCloudDemo-scene from KinectDemos/PointCloudDemo-folder.
7. If you still can’t see the point cloud on screen, select the VfxPointCloud-object and then press ‘Edit’ in the user interface of the ‘Visual Effect’-component. Then just close the VFX editor. This will update the visual effect asset, if needed.
8. If you want to filter only the players in the scene, please use 'Point cloud player list'-setting of the sensor interface, to set the needed player index, a list of player indices or all players (-1). The 'Get body frames'-setting of KinectManager should NOT be 'None', and 'Sync body and depth' should be enabled, to provide synchronization between the depth and body index images.

The Kinect Manager

All sensor-related scenes require a KinectManager-component. In all demo scenes it resides in the KinectController-game object. This is the most basic component that manages the sensors and data. Its public API is utilized by all other sensor-related components. You can also utilize the public API of KinectManager in your scripts. Please note, the KinectManager persists across the scenes. In case of multi-scene setup, it should be created only once – in the startup scene. All other scenes can always reference it by calling the static 'KinectManager.Instance'-property.

Here are the public settings of the KinectManager-component:

Setting:	***Description:***
Sensor Data	
Use Multi Cam Config	Whether to create the sensor interfaces according to the multi-camera config, if one is available. Otherwise use the sensor interfaces, as configured in the child objects of this object.
Sync Multi Cam Frames	Whether to synchronize the received frames, in case of master/sub cameras.
Get Depth Frames	Whether to get depth frames from the sensor(s).
Get Color Frames	Whether to get color frames from the sensor(s).
Get Infrared Frames	Whether to get infrared frames from the sensor(s).
Get Pose Frames	Whether to get pose frames from the sensor(s).
Get Body Frames	Whether to get body frames from the body tracker.
Sync Depth and Color	Whether to synchronize depth and color frames.
Sync Body and Depth	Whether to synchronize body and depth frames.
User Detection	
Min User Distance	Minimum distance to user, in order to be considered for body processing. 0 means no minimum distance limitation.
Max User Distance	Maximum distance to user, in order to be considered for body processing. 0 means no maximum distance limitation.
Max Left Right Distance	Maximum left or right distance to user, in order to be considered for body processing. 0 means no left/right distance limitation.
Max Tracked Users	Maximum number of users, who may be tracked simultaneously. 0 means no limitation.
Show Allowed Users Only	Whether to display only the users within the allowed distances, or all users.
User Detection Order	How to assign users to player indices - by order of appearance, distance or left-to-right.
Ignore Inferred Joints	Whether to ignore the inferred joints, or consider them as tracked joints.
Ignore Z-Coordinates	Whether to ignore the Z-coordinates of the joints (for 2D-scenes) or not.
Bone Orientation Constraints	Whether to apply the bone orientation constraints.
Player Calibration Pose	Calibration pose required, to start tracking the respective user.
User Manager	User manager, used to track the users. KM creates one, if not set.
Gesture Detection	
Gesture Manager	Gesture manager, used to detect user gestures. KM creates one, if not set.
On-Screen Info	
Display Images	List of images to display on the screen.
Display Image Width	Single image width, as percent of the screen width.
Status Info Text	UI-Text to display status messages.
Console Log Messages	Whether to log the KinectManager info messages to the console or not.
Events	
On Depth Sensors Started	Fired when the depth sensors get started.

On Depth Sensors Stopped	Fired when the depth sensors get stopped.

Here is the online documentation regarding the public API of this component, as well.

Sensor Interfaces

The K4A-package supports multiple sensor setups, as well as several types of sensors by design (Azure Kinect, Kinect-v2 & RealSense D400). In all demo scenes the available sensor interfaces reside on child objects of the KinectController-game object. At start up, the KinectManager tries to find the sensor interfaces on these child objects. If nothing is found, it creates a single Kinect4Azure interface on the same game object. Feel free to duplicate, create or remove sensor-interface objects, according to your specific setup. You can also change the sensor-interface settings, to match your setup needs. Please note, the transform-component of each sensor-interface object represents the sensor's position and rotation in the scene.

Here are the common settings of all sensor-interface components:

Setting:	***Description:***
Device Streaming Mode	Device streaming mode, in means of connected sensor, recording or disabled.
Device Index	Index of the depth sensor in the list of currently connected sensors.
Recording File	Path to the recording file, if the streaming mode is PlayRecording.
Min Distance	Minimum distance in meters, used for creating the depth-related images.
Max Distance	Maximum distance in meters, used for creating the depth-related images.
Point Cloud Resolution	Resolution of the generated point-cloud textures.
Point Cloud Vertex Texture	Render texture, used for point-cloud vertex mapping. The texture resolution should match the depth or color image resolution.
Point Cloud Color Texture	Render texture, used for point-cloud color mapping. The texture resolution should match the depth or color image resolution.
Point Cloud Player List	List of comma-separated player indices to be included in the point cloud. Use -1 for all players, or empty list for full point cloud.

General-Purpose Components

Here you can find the short descriptions of the general-purpose components. For detailed description of each component and its properties, select the component on the left (*this will be added later*). Please note, all Kinect-related components need the KinectManager-component in the scene. The KinectManager should persist across all scenes. To avoid having multiple KinectManager instances in the scene, see 'Howto-Use-KinectManager-Across-Multiple-Scenes.pdf'-manual in the _Readme-folder of the K4A-asset.

Component:	***Description:***
AvatarController	Avatar controller is the component that transfers the captured user motion to a humanoid model (avatar).
AvatarControllerClassic	Avatar controller classic allows manual assignment of model's rigged bones to the tracked body joints.
AvatarScaler	Avatar scaler is the component that scales avatar's body, according to the user's body and bone sizes.
BackgroundColorCamDepthImage	BackgroundColorCamDepthImage is component that displays the color camera aligned depth image on RawImage texture, usually the scene background.

BackgroundColorCamInfraredImage	BackgroundColorCamInfraredImage is component that displays the color camera aligned infrared image on RawImage texture, usually the scene background.
BackgroundColorCamUserImage	BackgroundColorCamUserImage is component that displays the color camera aligned user-body image on RawImage texture, usually the scene background.
BackgroundColorImage	Background color image is component that displays the color camera feed on RawImage texture, usually the scene background.
BackgroundDepthCamColorImage	Background depth-cam color image is component that displays the depth camera aligned color image on RawImage texture, usually the scene background.
BackgroundDepthImage	Background depth image is component that displays the depth camera image on RawImage texture, usually the scene background.
BackgroundInfraredImage	Background infrared image is component that displays the infrared camera image on RawImage texture, usually the scene background.
BackgroundRemovalByBodyBounds	BackgroundRemovalByBodyBounds filters user silhouettes, according to the bounds determined by the positions of the body joints.
BackgroundRemovalByBodyIndex	BackgroundRemovalByBodyIndex filters user silhouettes, according to the body index frames coming from the body tracking SDK.
BackgroundRemovalByDist	BackgroundRemovalByDist filters part of the real environment, according to the given spatial limits.
BackgroundRemovalByGreenScreen	BackgroundRemovalByGreenScreen filters color camera data, according to its similarity or difference to the color of the green-screen.
BackgroundRemovalManager	Background removal manager is the component that filters and renders user body silhouettes.
BackgroundStaticImage	Background static image is component that displays the static image on RawImage texture, usually the scene background.
BackgroundUserBodyImage	Background user-body image is component that displays the user-body image on RawImage texture, usually the scene background.
BodyDataRecorderPlayer	BodyDataRecorderPlayer is the component that can be used for recording and replaying of body-data files.
BodySlicer	Body slicer is component that estimates the user height from the raw depth data, as well as several other body sizes.
CubemanController	Cubeman controller transfers the captured user motion to a cubeman model.
DepthIrFilterImage	DepthIrFilterImage filters the sensor's IR image with the raw depth. The resulting image is displayed on the given RawImage.
FollowSensorTransform	Follow sensor transform makes the game object follow the position and rotation of the sensor.
FollowUserJointPose	FollowUserJointPose makes the game object's transform follow the given user's joint pose.
ForegroundBlendRenderer	ForegroundBlendRenderer provides volumetric rendering and lighting of the real environment, filtered by the background-removal manager.
ForegroundToRawImage	ForegroundToRawImage sets the texture of the RawImage-component to be the BRM's foreground texture.
ForegroundToRenderer	ForegroundToRenderer sets the texture of the Renderer-component to be the BRM's foreground texture.
HmdHeadMover	HmdHeadMover moves the avatar model, according to the camera position reported by the HMD tracker.
InteractionInputModule	InteractionInputModule is the input module that can be used as component of the Unity-UI EventSystem.
InteractionManager	InteractionManager is the component that controls the hand cursor and manages the hand interactions.
KinectEventManager	KinectEventManager provides sensor-frame events to the registered Unity event

	listeners.
KinectGestureManager	Kinect gesture manager is the component that tracks and processes the user gestures.
KinectManager	KinectManager is the the main and most basic depth-sensor related component. It controls the sensors and manages the data streams.
KinectNetServer	Kinect net-server listens for client connections and sends the data from the given sensor's streams to its clients over the network.
KinectUserBodyMerger	KinectUserBodyMerger merges user bodies detected by multiple connected sensors.
KinectUserManager	Kinect user manager is the component that tracks the users in front of the sensor.
MultiCameraSyncher	MultiCameraSyncher enables synchronization of frames between the master and the subordinate sensors.
PhotoShooter	Photo shooter provides public functions to take pictures of the currently rendered scene.
PointCloudTarget	PointCloudTarget sets the point cloud resolution of the respective sensor, as well as the target render textures.
PortraitBackground	Portrait background is the component that sets the background image in the required screen resolution.
PoseModelHelper	Pose model helper matches the sensor-tracked joints to model transforms.
PoseModelHelperClassic	Pose model helper matches the sensor-tracked joints to model transforms (with manual assignment).
SceneBlendRenderer	SceneBlendRenderer provides volumetric rendering and lighting of the real environment, as seen by the sensor's color camera.
SceneMeshRenderer	SceneMeshRenderer renders virtual mesh of the environment in the scene, as detected by the given sensor.
SceneMeshRendererGpu	SceneMeshRendererGpu renders virtual mesh of the environment in the scene, as detected by the given sensor. This component uses GPU for mesh processing rather than CPU.
UserMeshRenderer	UserMeshRenderer renders virtual mesh of a user in the scene, as detected by the given sensor.
UserMeshRendererGpu	UserMeshRendererGpu renders virtual mesh of a user in the scene, as detected by the given sensor. This component uses GPU for mesh processing rather than CPU.
UserSkeletonCollider	UserSkeletonCollider creates colliders for the joints and bones of the given user, so they can interact with the physical objects in the scene.

Demo-Specific Script Components

Here you can find the short descriptions of the demo-specific components. They are grouped in their respective demo-group folders. The demo-specific script component usually stays in the Scripts-subfolder of the respective demo-group folder.

(to be added later)

What to Redistribute

Here is how to reuse the Kinect-related scripts and components in your Unity project:

1. Copy folder ‘KinectScripts’ from the AzureKinectExamples-folder of this package to your project. This folder contains all needed components, scripts, filters and interfaces.
2. Copy folder ‘Resources’ from the AzureKinectExamples-folder of this package to your project. This folder contains some needed libraries and resources.
3. Copy the sensor-SDK specific sub-folders (Kinect4AzureSDK, KinectSDK2.0 & RealSenseSDK2.0) from the AzureKinectExamples/SDK-folder of this package to your project. It contains the plugins and wrapper classes for the respective sensor types.
4. Wait until Unity detects, imports and compiles the newly detected resources and scripts.
5. Please do not share the KinectDemos-folder in source form or as part of public repositories.