# Developer's Guide: ArcGIS Runtime SDK for Android – AR/VR beta 2

This guide describes how to use the ArcGIS Runtime SDK for Android to create native apps with augmented or virtual reality experiences. Support for these experiences is built on 3D functionality included in ArcGIS Runtime.

Augmented reality (AR) is a technology that superimposes computer-generated content on a live direct or indirect view of the real world. For ArcGIS Runtime, AR is achieved by rendering 3D GIS "computer-generated" or virtual content in a SceneView control on top of a transparent background where a view of the "real world" is displayed.

Virtual reality (VR) is a computer technology that uses virtual reality headsets to generate realistic images, sounds and other sensations to simulate a user's presence in a virtual or imaginary environment.
For ArcGIS Runtime, VR is achieved by enabling a stereoscopic view of 3D GIS "computer-generated" content in a SceneView control for use in a virtual reality device/headset.

The remainder of this guide will discuss how to install, enable, and utilize AR and VR behavior in ArcGIS Runtime.

## System Requirements

In general, system requirements for the private beta of the ArcGIS Runtime SDK for Android follow those defined on the ArcGIS for Developers website: https://developers.arcgis.com/android/latest/guide/system-requirements.htm. Notable exceptions are listed below:

- Minimum Android version: 4.4
- Minimum Android Studio version: 3.0

For devices or platforms that utilize specific sensors or frameworks to support AR or VR experiences, see the device or platform vendor for system requirements.

## Installation

The private beta of the ArcGIS Runtime SDK for Android is provided as a ".pom" and ".aar" files. Use the following steps to download, extract and reference:

1. Download the ArcGIS Runtime SDK for Android beta package (zip) from the Early Adopter site. The package is a zip file which contains the ".pom" and ".aar" files inside a "Lib" folder.
2. Extract the contents of the archive to a location on disk.
3. Copy the ".pom" and ".aar" files to the following location on your disk
   mac: /Users/[user-name]/.m2/repository/com/esri/arcgisruntime/arcgis-android/100.2.1-arbeta2/
4. Your full directory path should resemble the following:
   /Users/[user-name]/.m2/repository/com/esri/arcgisruntime/arcgis-android/100.2.1-arbeta2/arcgis-android-100.2.1-arbeta2.aar
   /Users/[user-name]/.m2/repository/com/esri/arcgisruntime/arcgis-android/100.2.1-arbeta2/arcgis-android-100.2.1-arbeta2.pom
5. In Android Studio, edit your build.gradle file to look at your local maven repository. For this add the following to your project root build.gradle file:

```
allprojects {
    repositories {
    mavenLocal()
    }
}
```

6. Add the following dependency to your app's build.gradle file (not the root build.gradle file this time):

```
dependencies {
```

```
...
implementation 'com.esri.arcgisruntime:arcgis-android:100.2.1-arbeta2'
}
```

## Augmented Reality

Support for building a native augmented reality experience using ArcGIS Runtime depends on a number of factors associated with device capabilities and data accuracy. These factors include, but are not limited to:

- Enabling display of virtual content on a view of the real world. Proper spatial association of the virtual content and physical (ie real) world depends on a common coordinate system, the accuracy of the virtual data (ie layers), and the reported location of the device on which the virtual and physical views are overlaid. Orienting the field of view for both the virtual content in a scene and video representation of the real world is essential to proper interaction within the augmented reality experience.
- Synchronizing interaction with virtual content and the real world. To put it another way, through the display on a device a user should be able to interact with virtual content as if it exists in the real world. This requires position and orientation information from a device via its sensors. Some devices have sensors and frameworks that provide more accurate/resolute location and proximity details than others.

### Getting Started

Consider the following steps to enable display and interaction with virtual GIS content on a view of the real world:

1) To represent/view the real world, display the video feed from a device camera. The SceneView control will be displayed on top of the camera view to provide a composite view (virtual and real content). Support for using the video feed from a camera does not require Esri or ArcGIS Runtime technology. In short, the development platform provides access to this device capability/sensor. See AR oriented samples associated with this product for techniques on display of a camera feed. We do provide such a class in our sample app part of the beta zip file.

```
import com.esri.arcgisruntime.mapping.view.SceneView;
(...)
private SceneView mSceneView;
```

2) Create a SceneView and add 3D content (eg layers, graphics overlays) you intend to overlay on a video feed of the physical/real world.

```
import com.esri.arcgisruntime.mapping.ArcGISScene;
(...)
mSceneView.setScene(new ArcGISScene());
mSceneView.getScene().getOperationalLayers().add(<operational layer>);
```

3) Set the SceneView ARModeEnabled property to true. This will set the SceneView background to transparent, hide the background grid, and turn off atmospheric effects.

```
mSceneView.setARModeEnabled(true);
```

4) Define how position and orientation of the device will enable the user to interact with virtual content. In general, this requires integration with device sensors. Virtual content is often presented at full/room scale (same scale as real world) or small/model scale (smaller than real world). In either case, the location of the virtual content is associated with real world coordinates, and control of the SceneView camera needs to synchronize with the location and orientation of the device camera. ArcGIS Runtime provides a FirstPersonCameraController to manage the SceneView camera using device sensors. The

FirstPersonCameraController needs an initial position to determine where to begin interaction with the virtual content. At full scale, initial position is usually the current geographic location (x,y,z) and orientation (heading, pitch, roll) of the device. At small scale, it's often a position within the virtual content (as is the case with the example code below):

```
import com.esri.arcgisruntime.mapping.view.Camera;
import com.esri.arcgisruntime.mapping.view.FirstPersonCameraController;
(...)
Camera cameraSanDiego = new Camera(32.707, -117.157, 60, 180, 0, 0);
FirstPersonCameraController fpcController = new FirstPersonCameraController();
fpcController.setInitialPosition(cameraSanDiego);
```

The FirstPersonCameraController exposes a set of properties that are essential to enabling interaction with the scene view. The most important property and type is DeviceMotionDataSource. This is an abstract class that provides the framework for registering platform specific device sensors with the scene view's camera controller. In general, implementation of device motion data sources is where most of the custom AR development work will take place. *See the **Custom device motion data sources** section below for details on code examples.* A set of pre-compiled device motion data sources are provided with the API to enable integration with common sensor APIs or platform specific frameworks, such as ARKit or ARCore (future). The PhoneMotionDataSource class is provided for Android as it integrates with hardware sensors available in this platform devices. These basic sensor will provide orientation and location information, but do not accurately account for movement, so initial location remains static. A set of methods enable use of orientation (StartUpdatingAngles) or location (StartUpdatingInitialLocation) or both (StartAll). StartUpdatingAngles is important to enable a user to move the device. In general, StartUpdatingInitialLocation is important if the location of the device is important to rendering the virtual content. For example, if the device is on a worksite and virtual content is a structure to be built on that site.

```
import com.esri.arcgisruntime.mapping.view.PhoneMotionDataSource;
(...)
PhoneMotionDataSource phoneSensors = new PhoneMotionDataSource(this);
fpcController.setDeviceMotionDataSource(phoneSensors);
mSceneView.setCameraController(fpcController);

// To update position and orientation of the camera with device sensors use:
phoneSensors.startUpdatingAngles(false);
// To update location of camera in the scene with device location (GPS) use:
//phoneSensors.startUpdatingInitialPosition();
// To update both use:
//phoneSensors.startAll();
```

Another option will be the ARCoreMotionDataSource which will enables integration with Android devices that will support ARCore functionalities when they get officially released. Fortunately this framework will provides reasonably accurate feedback on relative device location and movement, so it enables users to navigate through a synchronized view of virtual and real-world content.

This works well for AR experiences where the virtual and real world objects are presented at the same scale. However, there are situations where the virtual content in the scene view is displayed at a different scale than the real world, such as display of a small scale, virtual model of a building on a table top. In that situation, changes in location of the user/device in the real world should translate to reasonable movement within the scene view to navigate virtual content appropriately. To accommodate, the TranslationFactor property defines the number of units in the virtual world equal to one unit in the real world. For example, if this property is set to 100, the scene view camera will move 100 meters in virtual space when the user/device moves 1 meter in the real world.

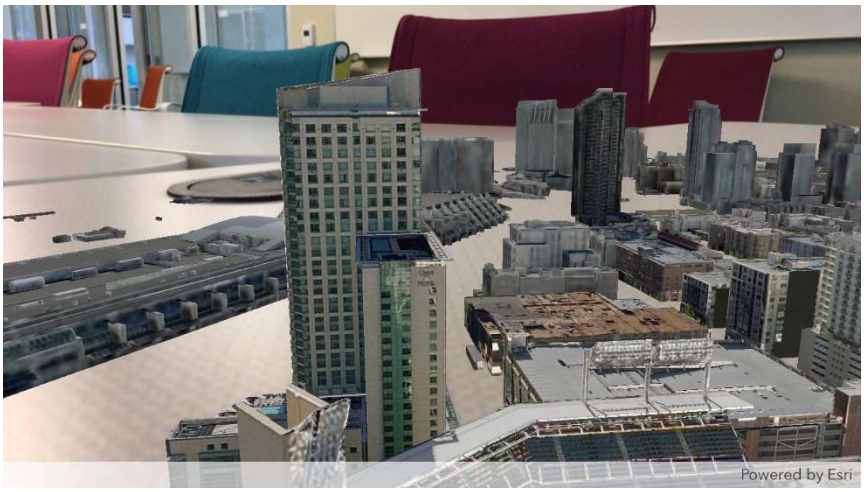
```
fpcController.setTranslationFactor(100);
```

5) Refine the display of virtual content to accommodate for user needs. In general, this involves determining the optimum framerate to provide appropriate interaction with virtual content overlaid on the real world. For example, virtual objects should remain anchored to real world positions when moving a device. In addition, quality of the virtual objects needs to be adequate to be useful. In the Runtime API, the FirstPersonCameraController has a Framerate property to manage this behavior.

```
fpcController.setFramerate(FirstPersonCameraController.FirstPersonFramerate.BALANCED);
```

Three options are available: Quality, Balanced, and Speed. Quality retains the data loading pattern available by default in a scene view where data, at full fidelity, is loaded each time the scene view camera (and device) moves. This is ideal for devices with higher end processors and higher memory availability. Balanced changes the data loading pattern in two ways. First, data will be loaded around the scene view camera position, not just want is in the view. This assumes the device will change position and orientation quickly. And second, scene layers will load the same level of detail. This means less data and smaller memory footprint. This option is best for AR experiences on mobile devices. Speed will match the data loading patter of Balanced, but the amount of data loaded around the scene view camera will be dictated by what the GPU, not local memory. Ideally this option enables data display at 60 frames per second and offers an ideal solution for VR experiences.

Screenshot of AR in ArcGIS Runtime SDK for Android – AR/VR beta 2




Screenshot of AR in ArcGIS Runtime SDK for Android – AR/VR beta 2


## Virtual Reality

Support for building a native virtual reality experience using ArcGIS Runtime depends on a number of factors associated with device capabilities and interactive experiences. These factors include, but are not limited to:

- Enabling display of virtual content to replace and/or simulate the real world. This functionality is founded on basic optical display properties that enable an immersive 3D experience through stereo rendering – in short, two images slightly offset and distorted to exhibit perception of depth when viewed with two eyes. In general, visually a user is teleported to another location represented by the images. When a user's entire field of view is replaced with a virtual world (eg. as it is with VR headsets) the virtual experience will be more immersive. Of course, quality of data and resolution of the display will impact the fidelity (realistic nature) of the virtual experience.
- Synchronizing interaction between the device/user and virtual world. To move or interact within the virtual world usually requires user action. In general, device sensors and peripherals (eg controllers) can provide inputs to change camera position or location in a virtual world. This movement must feel natural to be useful, which means synchronizing movements of the device (real world) with the same movement in the virtual world and/or effectively transitioning between locations in the virtual world. Technically this requires a mechanism to interact with device inputs, an API with properly timed and responsive display behavior, and a device with resources to handle content demands in a virtual experience.

### Getting Started

Consider the following steps to enable display and interaction with virtual GIS content in a virtual world:

1) Contents of the SceneView will simulate\replace the real world. Create a SceneView and add a basemap for context and 3D content as operational data. Using an elevation source will often provide a better, more realistic experience.

2)
```
import com.esri.arcgisruntime.mapping.ArcGISScene;
(...)
mSceneView.setScene(new ArcGISScene(Basemap.Type.IMAGERY));
mSceneView.getScene().getOperationalLayers().add(<operational layer>);
mSceneView.getScene().getBaseSurface().getElevationSources().add(<elevation layer>);
```

3) Set the SceneView.StereoRendering property to define the virtual experience. In the current release, only a side by side, barrel distorted stereoscopic display is supported (SideBySideBarrelDistortionStereoRendering). This rendering mode is designed to support VR experience with headsets, such that provided by Google Cardboard and Samsung Gear VR.

```
import com.esri.arcgisruntime.mapping.view.SideBySideBarrelDistortionStereoRendering;
(...)
mSceneView.setStereoRendering(new SideBySideBarrelDistortionStereoRendering());
```

4) Define how position and orientation of the device will enable the user to interact with the virtual world. In general, this requires integration with device sensors and peripherals. ArcGIS Runtime provides a FirstPersonCameraController to manage the SceneView camera using device sensors. The FirstPersonCameraController needs an initial position to determine where to begin interaction with the virtual world. Initial position in the virtual world defined by geographic location (x,y,z) and camera orientation (heading, pitch, roll) usually associated with the device orientation.

```
import com.esri.arcgisruntime.mapping.view.Camera;
import com.esri.arcgisruntime.mapping.view.FirstPersonCameraController;
(...)
Camera cameraSanDiego = new Camera(32.707, -117.157, 60, 180, 0, 0);
FirstPersonCameraController fpcController = new FirstPersonCameraController();
fpcController.setInitialPosition(cameraSanDiego);
```

The FirstPersonCameraController exposes a set of properties that are essential to enabling interaction with the scene view. The most important property and type is DeviceMotionDataSource. This is an abstract class that provides the framework for registering platform specific device sensors with the scene view's camera controller. In general, implementation of device motion data sources is where most of the custom VR development work will take place. *See the **Custom device motion data sources** section below for details.* The PhoneMotionDataSource class is provided for Android as it integrates with hardware sensors available in its devices. These basic sensors will provide orientation information (so you can look around the scene), but do not accurately account for movement, so location in the virtual world does not change based on relative device position. In most cases, VR experiences will not use the device location, only the orientation to match movements between the real and virtual worlds. In the following code example, the call to StartUpdatingAngles enables synchronization of orientation between device and virtual world.

```
import com.esri.arcgisruntime.mapping.view.PhoneMotionDataSource;
(...)
PhoneMotionDataSource phoneSensors = new PhoneMotionDataSource(this);
fpcController.setDeviceMotionDataSource(phoneSensors);
mSceneView.setCameraController(fpcController);

// To update position and orientation of the camera with device sensors use:
phoneSensors.startUpdatingAngles(false);
// To update location of camera in the scene with device location (GPS) use:
//phoneSensors.startUpdatingInitialPosition();
```

```
// To update both use:
//phoneSensors.startAll();
```

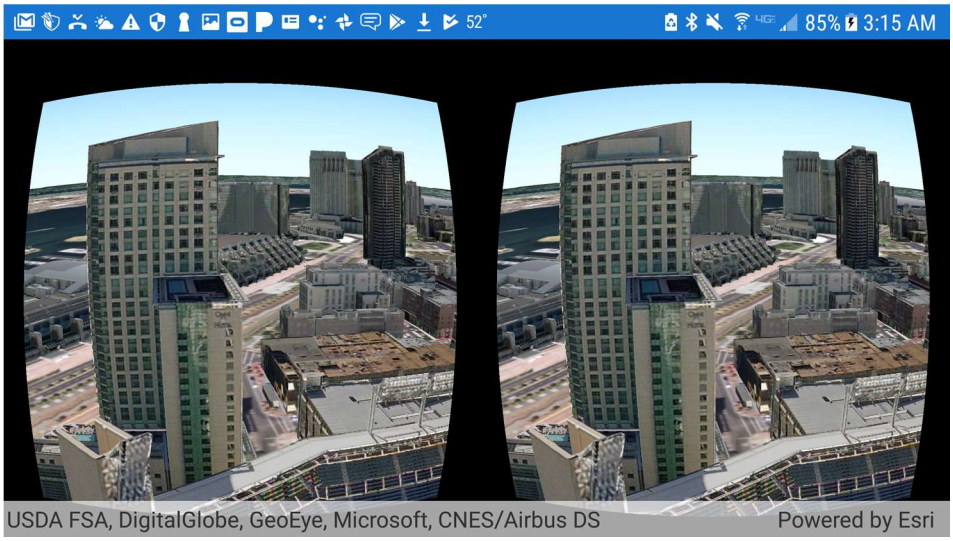
Another option will be the **ARCoreMotionDataSource** which will enable integration with Android devices that supports it when it gets released. While built to support AR experiences, the framework actually provides usable feedback on relative device location and orientation, so it enables movement of the device to be more responsive and accurate. Although many VR experiences using a headset do not require or encourage the user to change position in the real world, some may permit limited movement within a safe area. Use of this framework may provide a better option for tracking that movement.

5) Refine the display of the virtual world to accommodate for user needs. In general, this involves determining the optimum framerate to provide responsive interaction in the virtual experience. In the Runtime API, the **FirstPersonCameraController** has a **Framerate** property to manage this behavior.

```
fpcController.setFramerate(FirstPersonCameraController.FirstPersonFramerate.SPEED);
```

Three options are available: Quality, Balanced, and Speed. **Quality** retains the data loading pattern available by default in a scene view where data, at full fidelity, is loaded each time the scene view camera (and device) moves. This is ideal for devices with higher end processors and higher memory availability. **Balanced** changes the data loading pattern in two ways. First, data will be loaded around the scene view camera position, not just what is in the view. This assumes the device will change position and orientation quickly. And second, scene layers will load the same level of detail. This means less data and smaller memory footprint. This option is best for AR experiences on mobile devices. **Speed** will match the data loading patter of Balanced, but the amount of data loaded around the scene view camera will be dictated by the GPU, not local memory. Ideally this option enables data display at 60 frames per second and offers an ideal solution for VR experiences.

Screenshot of VR in **ArcGIS Runtime SDK for Android – AR/VR beta 2**



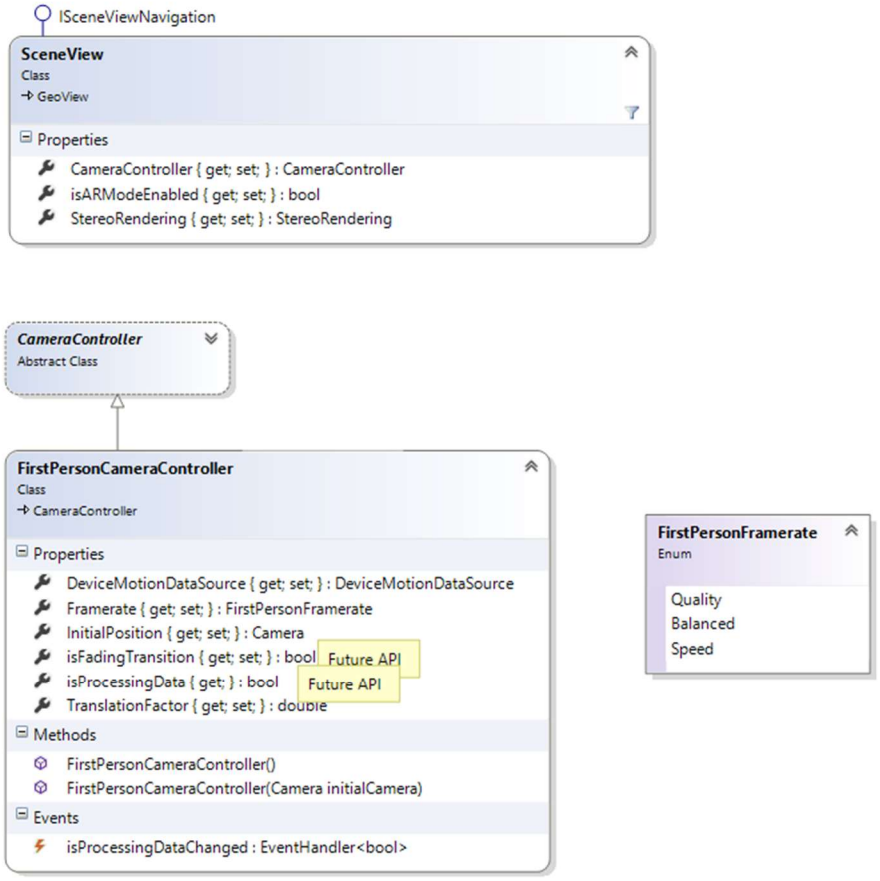

## Custom device motion data sources

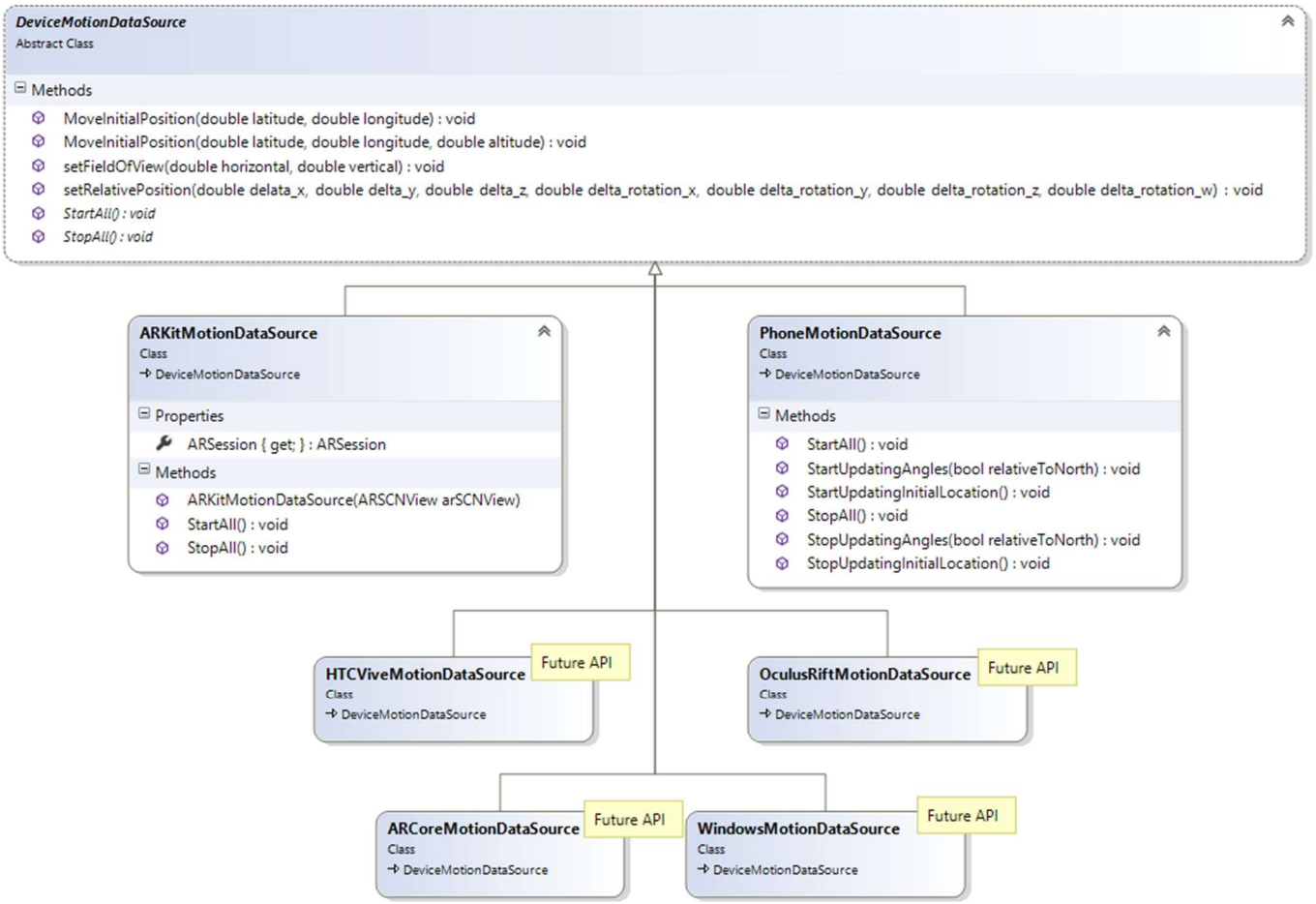
As devices offer different sensors and capabilities to orient and position themselves in the real world, custom device motion data sources can be created to take advantage of these capabilities in AR and VR experiences built with

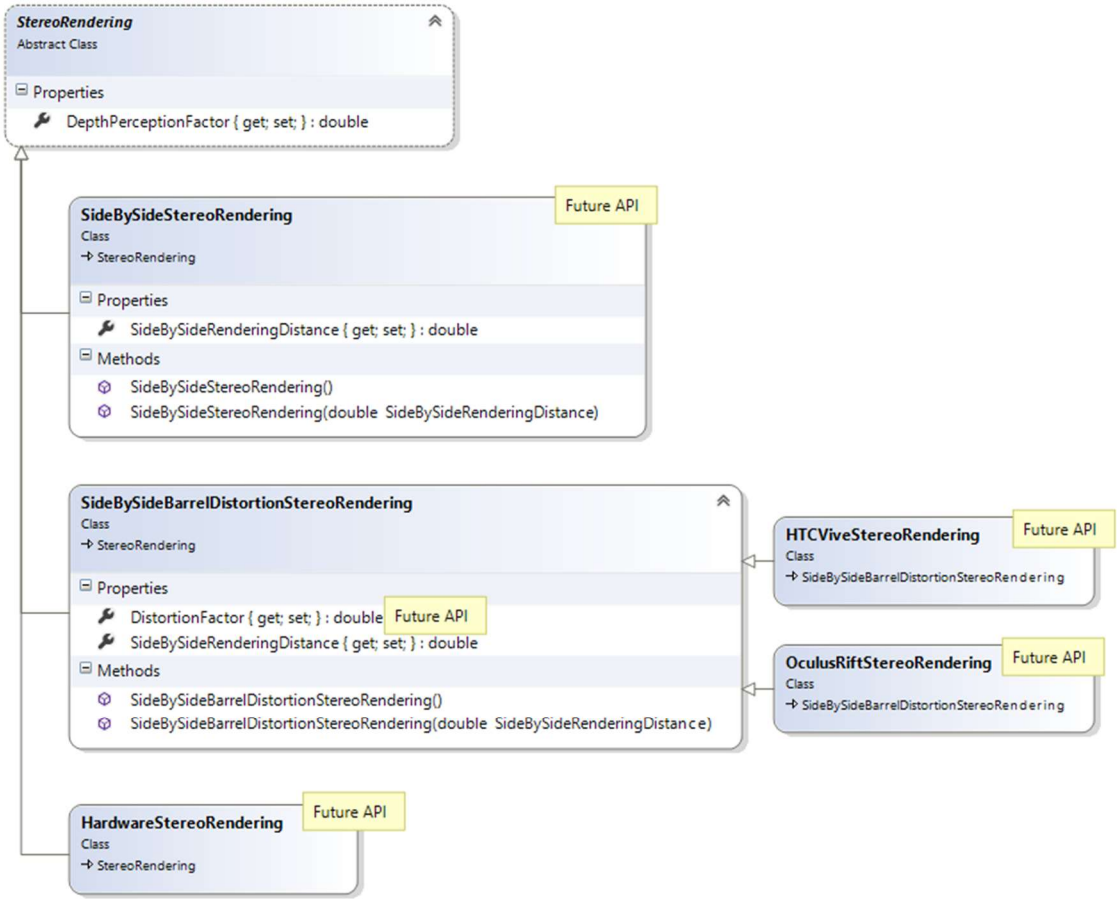
ArcGIS Runtime. Currently the **PhoneMotionDataSource** (and **ARCoreMotionDataSource** in the future**)** is included with the ArcGIS Runtime SDK. The API diagram section below lists a couple additional examples that can be introduced in the future.

## API Diagram

The following diagram shows components within the ArcGIS Runtime API which are utilized in AR or VR experiences. Some of the components exist in the current beta release. Others are considered future work for the API (as marked), but demonstrate where and how the Runtime will be enhanced to support AR and VR frameworks and device types.




DeviceMotionDataSource
Abstract Class
Methods
MoveInitialPosition(double latitude, double longitude) : void
MoveInitialPosition(double latitude, double longitude, double altitude) : void
setFieldOfView(double horizontal, double vertical) : void
setRelativePosition(double delata_x, double delta_y, double delta_z, double delta_rotation_x, double delta_rotation_y, double delta_rotation_z, double delta_rotation_w) : void
StartAll() : void
StopAll() : void
ARKitMotionDataSource
Class
DeviceMotionDataSource
Properties
ARSession { get; } : ARSession
Methods
ARKitMotionDataSource(ARSCNView arSCNView)
StartAll() : void
StopAll() : void
PhoneMotionDataSource
Class
DeviceMotionDataSource
Methods
StartAll() : void
StartUpdatingAngles(bool relativeToNorth) : void
StartUpdatingInitialLocation() : void
StopAll() : void
StopUpdatingAngles(bool relativeToNorth) : void
StopUpdatingInitialLocation() : void
HTCViveMotionDataSource
Future API
Class
DeviceMotionDataSource
OculusRiftMotionDataSource
Future API
Class
DeviceMotionDataSource
ARCoreMotionDataSource
Future API
Class
DeviceMotionDataSource
WindowsMotionDataSource
Future API
Class
DeviceMotionDataSource


StereoRendering
Abstract Class
Properties
DepthPerceptionFactor { get; set; } : double
SideBySideStereoRendering
Class
StereoRendering
Future API
Properties
SideBySideRenderingDistance { get; set; } : double
Methods
SideBySideStereoRendering()
SideBySideStereoRendering(double SideBySideRenderingDistance)
SideBySideBarrelDistortionStereoRendering
Class
StereoRendering
Properties
DistortionFactor { get; set; } : double
Future API
SideBySideRenderingDistance { get; set; } : double
Methods
SideBySideBarrelDistortionStereoRendering()
SideBySideBarrelDistortionStereoRendering(double SideBySideRenderingDistance)
HTCViveStereoRendering
Class
SideBySideBarrelDistortionStereoRendering
Future API
OculusRiftStereoRendering
Class
SideBySideBarrelDistortionStereoRendering
Future API
HardwareStereoRendering
Class
StereoRendering
Future API