AVFoundation

Set Up Capture Device

Requesting Authorization for Camera Usage

requestAccess(for:completionHandler:)

Requests the user's permission to allow the app to capture media of a particular type.

iOS 7.0+ | iPadOS 7.0+ | macOS 10.14+ | Mac Catalyst 14.0+ | tvOS 17.0+ | visionOS 1.0+

```
class func requestAccess(
    for mediaType: AVMediaType,
    completionHandler handler: @escaping (Bool) -> Void
)
```

Parameters

mediaType
A media type for which to check the authorization status. The supported media types are `video` and `audio`.

handler
A callback the system invokes with a Boolean value that indicates whether the user granted or denied access to your app.

Return control to the main queue or `MainActor` before performing user interface updates.

Discussion

Concurrency Note

You can call this method from synchronous code using a completion handler, as shown on this page, or you can call it as an asynchronous method that has the following declaration:

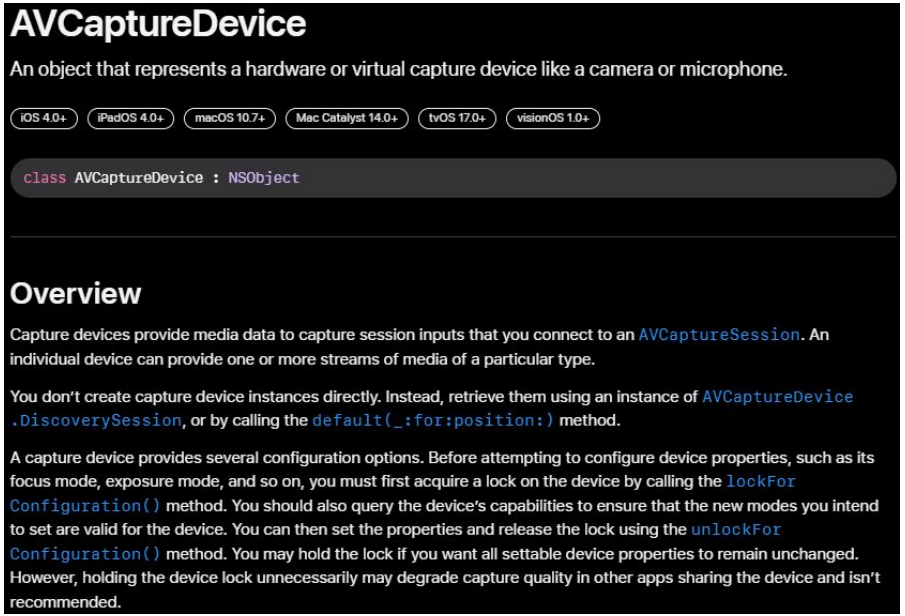
```
class func requestAccess(for mediaType: AVMediaType) async -> Bool
```

For information about concurrency and asynchronous code in Swift, see Calling Objective-C APIs Asynchronously.

Capturing media requires explicit permission from the user. An app's default authorization status is `AVAuthorizationStatus.notDetermined`, which means the user hasn't yet granted it permission to capture media. The first time you create an `AVCaptureDeviceInput` object for a media type that requires permission, the system automatically displays an alert to request recording permission. Alternatively, call this method to prompt the user at a time of your choosing. The system saves the user's selection so that it doesn't have to prompt the user again. A user can change their authorization status in the Settings app.

It seems we don't need to explicitly call requestAccess function in order to get permission from the user to get access to the specified device (in our case, the camera) since it says that “The first time you create an AVCaptureDeviceInput object for a media type that requires permission, the system automatically displays an alert to request recording permission.”

Capture Device Creation



AVCaptureDevice

An object that represents a hardware or virtual capture device like a camera or microphone.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+ | visionOS 1.0+

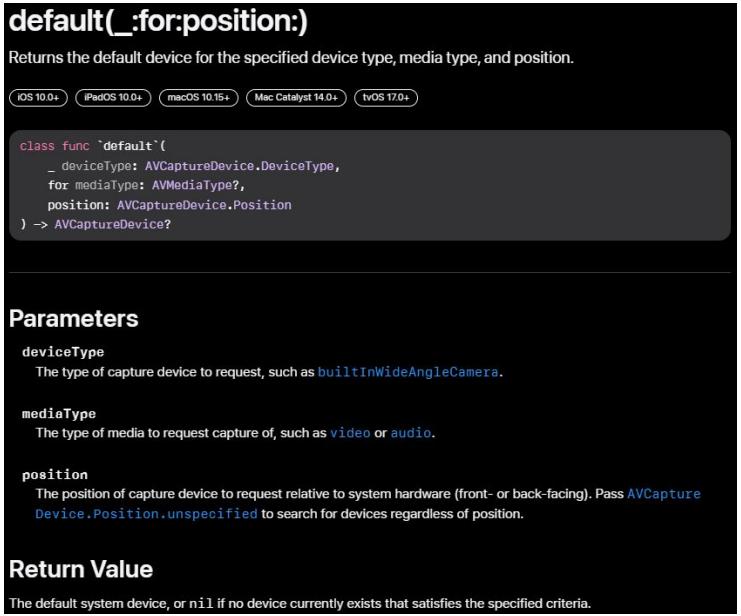
```
class AVCaptureDevice : NSObject
```

Overview

Capture devices provide media data to capture session inputs that you connect to an `AVCaptureSession`. An individual device can provide one or more streams of media of a particular type.

You don't create capture device instances directly. Instead, retrieve them using an instance of `AVCaptureDevice.DiscoverySession`, or by calling the `default(_:for:position:)` method.

A capture device provides several configuration options. Before attempting to configure device properties, such as its focus mode, exposure mode, and so on, you must first acquire a lock on the device by calling the `lockForConfiguration()` method. You should also query the device's capabilities to ensure that the new modes you intend to set are valid for the device. You can then set the properties and release the lock using the `unlockForConfiguration()` method. You may hold the lock if you want all settable device properties to remain unchanged. However, holding the device lock unnecessarily may degrade capture quality in other apps sharing the device and isn't recommended.



default(_:for:position:)

Returns the default device for the specified device type, media type, and position.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class func `default`(
    _ deviceType: AVCaptureDevice.DeviceType,
    for mediaType: AVMediaType?,
    position: AVCaptureDevice.Position
) -> AVCaptureDevice?
```

Parameters

deviceType
The type of capture device to request, such as `builtInWideAngleCamera`.

mediaType
The type of media to request capture of, such as `video` or `audio`.

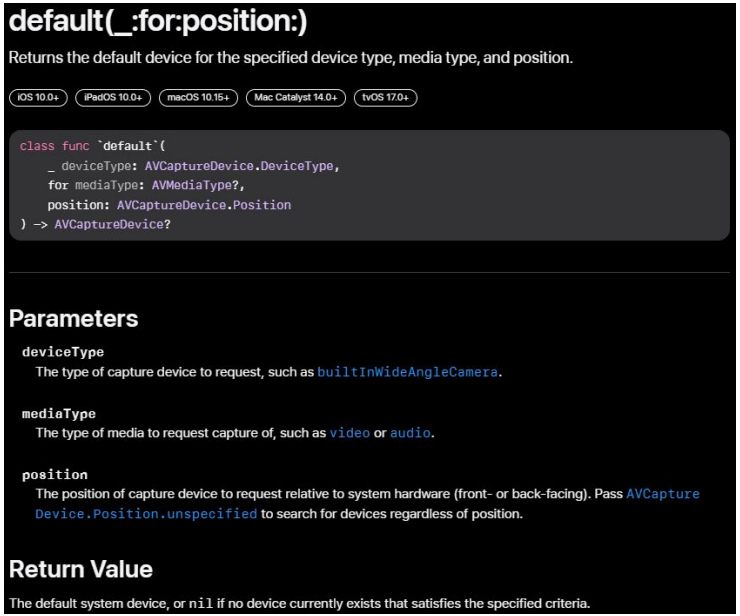
position
The position of capture device to request relative to system hardware (front- or back-facing). Pass `AVCaptureDevice.Position.unspecified` to search for devices regardless of position.

Return Value

The default system device, or `nil` if no device currently exists that satisfies the specified criteria.

Create the capture device first, and then with the instance of the device, invoke lockForConfiguration() and unlockForConfiguration() for configuring the instance.

Capture Device Creation



default(_:for:position:)

Returns the default device for the specified device type, media type, and position.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class func `default`(
    _ deviceType: AVCaptureDevice.DeviceType,
    for mediaType: AVMediaType?,
    position: AVCaptureDevice.Position
) -> AVCaptureDevice?
```

Parameters

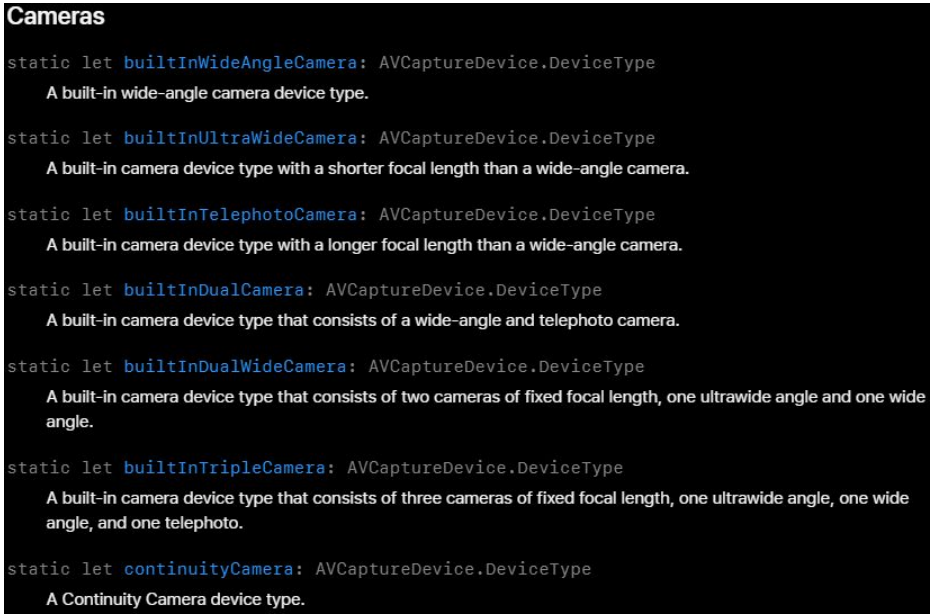
deviceType
The type of capture device to request, such as builtInWideAngleCamera.

mediaType
The type of media to request capture of, such as video or audio.

position
The position of capture device to request relative to system hardware (front- or back-facing). Pass AVCaptureDevice.Position.unspecified to search for devices regardless of position.

Return Value

The default system device, or nil if no device currently exists that satisfies the specified criteria.



Cameras

static let builtInWideAngleCamera: AVCaptureDevice.DeviceType
A built-in wide-angle camera device type.

static let builtInUltraWideCamera: AVCaptureDevice.DeviceType
A built-in camera device type with a shorter focal length than a wide-angle camera.

static let builtInTelephotoCamera: AVCaptureDevice.DeviceType
A built-in camera device type with a longer focal length than a wide-angle camera.

static let builtInDualCamera: AVCaptureDevice.DeviceType
A built-in camera device type that consists of a wide-angle and telephoto camera.

static let builtInDualWideCamera: AVCaptureDevice.DeviceType
A built-in camera device type that consists of two cameras of fixed focal length, one ultrawide angle and one wide angle.

static let builtInTripleCamera: AVCaptureDevice.DeviceType
A built-in camera device type that consists of three cameras of fixed focal length, one ultrawide angle, one wide angle, and one telephoto.

static let continuityCamera: AVCaptureDevice.DeviceType
A Continuity Camera device type.

For default creation of the device and for our purpose, choose from these device types (For our purpose, choose .builtInUltraWideCamera)

Capture Device Creation

Media Types

`static let audio: AVMediaType`
The media contains audio media.

`static let closedCaption: AVMediaType`
The media contains closed-caption content.

`static let depthData: AVMediaType`
The media contains depth data.

`static let haptic: AVMediaType`
The media contains haptic content.

`static let metadata: AVMediaType`
The media contains metadata.

`static let metadataObject: AVMediaType`
The media contains metadata objects.

`static let muxed: AVMediaType`
The media contains muxed media.

`static let subtitle: AVMediaType`
The media contains subtitles.

`static let text: AVMediaType`
The media contains text.

`static let timecode: AVMediaType`
The media contains a time code.

`static let video: AVMediaType`
The media contains video.

default(_:for:position:)

Returns the default device for the specified device type, media type, and position.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class func `default`(
    _ deviceType: AVCaptureDevice.DeviceType,
    for mediaType: AVMediaType?,
    position: AVCaptureDevice.Position
) -> AVCaptureDevice?
```

Parameters

deviceType
The type of capture device to request, such as `builtInWideAngleCamera`.

mediaType
The type of media to request capture of, such as `video` or `audio`.

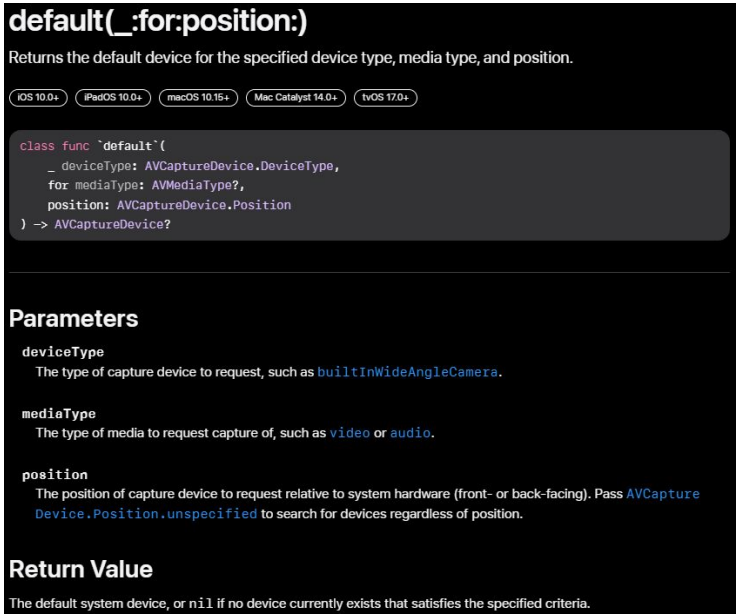
position
The position of capture device to request relative to system hardware (front- or back-facing). Pass `AVCaptureDevice.Position.unspecified` to search for devices regardless of position.

Return Value

The default system device, or `nil` if no device currently exists that satisfies the specified criteria.

For default creation of the device and for our purpose, choose from these media types (For our purpose, choose .video)

Capture Device Creation



default(_:for:position:)

Returns the default device for the specified device type, media type, and position.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class func `default`(
    _ deviceType: AVCaptureDevice.DeviceType,
    for mediaType: AVMediaType?,
    position: AVCaptureDevice.Position
) -> AVCaptureDevice?
```

Parameters

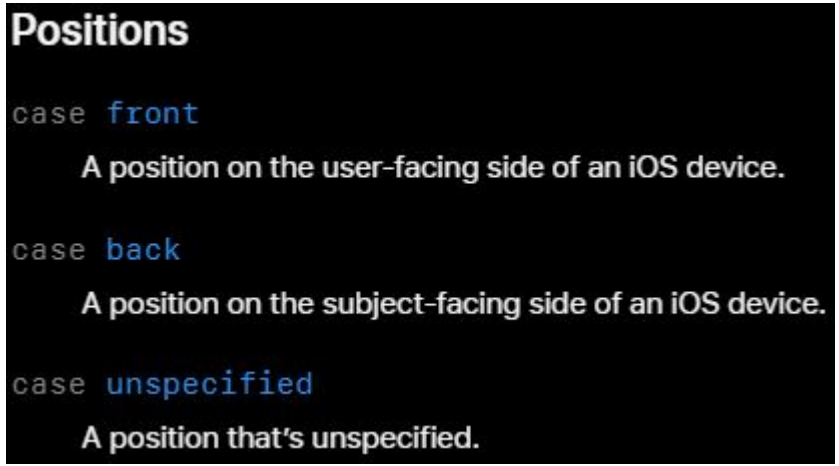
deviceType
The type of capture device to request, such as `builtInWideAngleCamera`.

mediaType
The type of media to request capture of, such as `video` or `audio`.

position
The position of capture device to request relative to system hardware (front- or back-facing). Pass `AVCaptureDevice.Position.unspecified` to search for devices regardless of position.

Return Value

The default system device, or `nil` if no device currently exists that satisfies the specified criteria.



Positions

`case front`

A position on the user-facing side of an iOS device.

`case back`

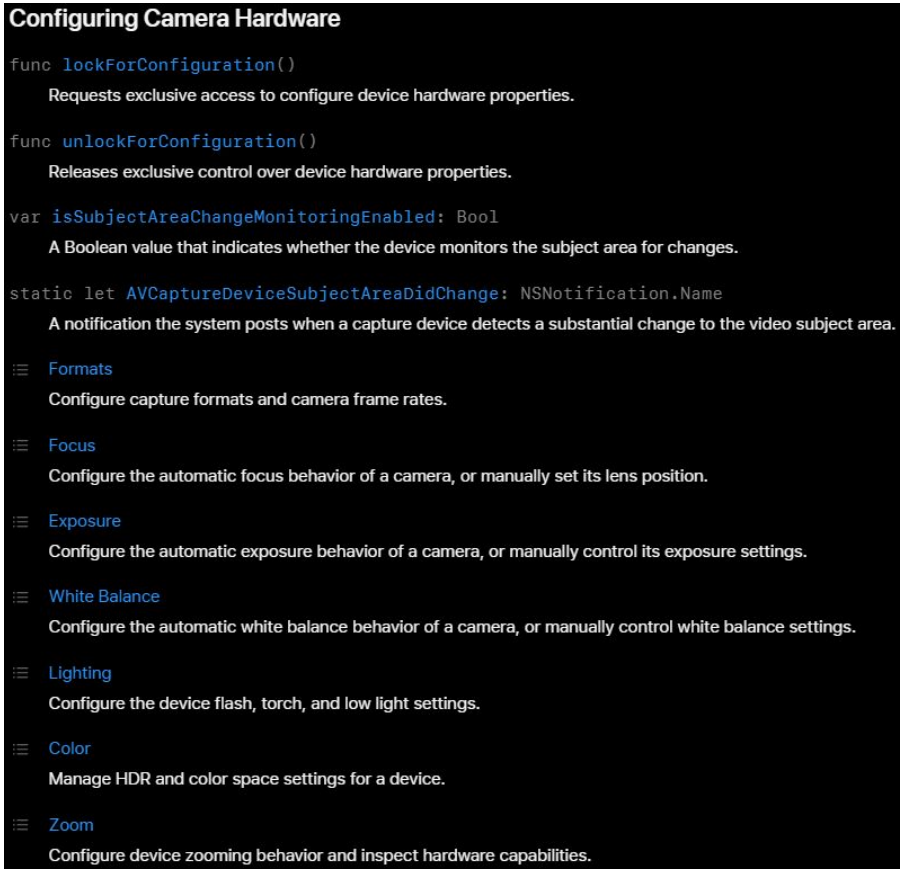
A position on the subject-facing side of an iOS device.

`case unspecified`

A position that's unspecified.

For default creation of the device and for our purpose, choose from these positions
(For our purpose, choose .back)

Configuring Capture Device



Use the functions lockForConfiguration() and unlockForConfiguration() before and after configuring the hardware (ie. The capture device created with the method explained in the previous slide).

Configuring Capture Device (Focus)

Configuring Automatic Focus

```
func isFocusModeSupported(AVCaptureDevice.FocusMode) -> Bool
```
Returns a Boolean value that indicates whether the device supports the specified focus mode.

```
var focusMode: AVCaptureDevice.FocusMode
```
The capture device's focus mode.

```
enum AVCaptureDevice.FocusMode
```
Constants to specify the focus mode of a capture device.

```
var isSmoothAutoFocusSupported: Bool
```
A Boolean value that indicates whether the device supports smooth autofocus.

```
var isSmoothAutoFocusEnabled: Bool
```
A Boolean value that indicates whether smooth autofocus is in an enabled state on the device.

```
var isFaceDrivenAutoFocusEnabled: Bool
```
A Boolean value that indicates whether the device has face-driven autofocus enabled.

```
var automaticallyAdjustsFaceDrivenAutoFocusEnabled: Bool
```
A Boolean value that indicates whether the device automatically adjusts face-driven autofocus.

```
var isAutoFocusRangeRestrictionSupported: Bool
```
A Boolean value that indicates whether the device supports focus range restrictions.

```
var autoFocusRangeRestriction: AVCaptureDevice.AutoFocusRangeRestriction
```
A value that controls the allowable range for automatic focusing.

```
enum AVCaptureDevice.AutoFocusRangeRestriction
```
Constants to specify the autofocus range of a capture device.

Setting a Focus Point of Interest

```
var isFocusPointOfInterestSupported: Bool
```
A Boolean value that indicates whether the device supports a point of interest for focus.

```
var focusPointOfInterest: CGPoint
```
The point of interest for focusing.

Setting Focus Manually

```
var isLockingFocusWithCustomLensPositionSupported: Bool
```
A Boolean value that indicates whether the device supports locking focus to a specific lens position.

```
var lensPosition: Float
```
The current focus position of the lens.

```
class let currentLensPosition: Float
```
A constant that represents the current lens position.

```
func setFocusModeLocked(lensPosition: Float, completionHandler: ((CMTime) -> Void)?)
```
Locks the lens position at the specified value, and sets the focus mode to a locked state.

Configuring Capture Device (Focus)

isFaceDrivenAutoFocusEnabled

A Boolean value that indicates whether the device has face-driven autofocus enabled.

iOS 15.4+ | iPadOS 15.4+ | Mac Catalyst 15.4+ | tvOS 17.0+

```
var isFaceDrivenAutoFocusEnabled: Bool { get set }
```

Discussion

Face-driven auto focus takes a subject's face into account when adjusting auto focus. For apps that link against iOS 15.4 or later, the value of this property defaults to `true` for devices that support auto focus.

Before setting a value for this property, perform the following:

- Obtain exclusive access to the device by calling its `lockForConfiguration()` method.
- Set the value of the device's `automaticallyAdjustsFaceDrivenAutoFocusEnabled` property to `false`.

Attempting to set a value before performing these steps results in an exception.

When you finish configuring the device, unlock it by calling its `unlockForConfiguration()` method.

Important

Updating the state of this property doesn't initiate a focus change. After setting a new value, set an appropriate `focusMode` to apply the change.

automaticallyAdjustsFaceDrivenAutoFocusEnabled

A Boolean value that indicates whether the device automatically adjusts face-driven autofocus.

iOS 15.4+ | iPadOS 15.4+ | Mac Catalyst 15.4+ | tvOS 17.0+

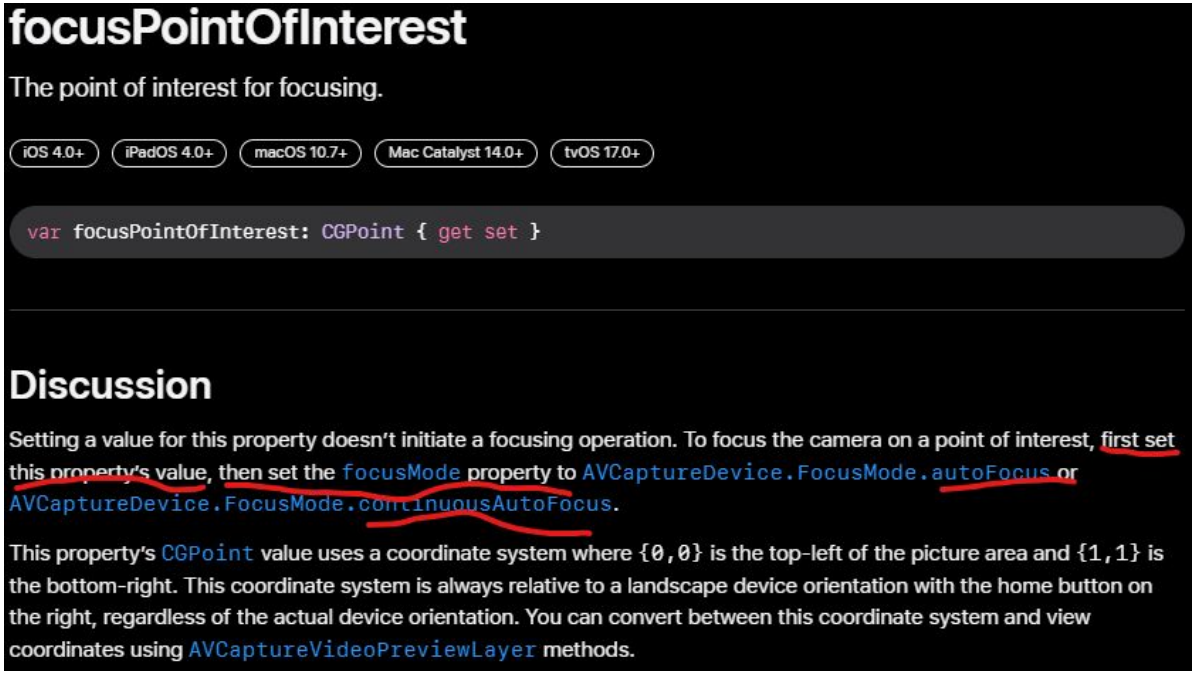
```
var automaticallyAdjustsFaceDrivenAutoFocusEnabled: Bool { get set }
```

Discussion

The value of this property defaults to `true` for devices that support auto focus. If your app requires explicitly setting the state of `isFaceDrivenAutoFocusEnabled`, set this value to `false`.

To set this property value, you must call the device's `lockForConfiguration()` method to obtain exclusive access to configure it. Otherwise, attempting to set a value raises an exception. When you finish configuring the device, call `unlockForConfiguration()` to release the lock.

Configuring Capture Device (Focus)



focusPointOfInterest

The point of interest for focusing.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var focusPointOfInterest: CGPoint { get set }
```

Discussion

Setting a value for this property doesn't initiate a focusing operation. To focus the camera on a point of interest, first set this property's value, then set the `focusMode` property to `AVCaptureDevice.FocusMode.autoFocus` or `AVCaptureDevice.FocusMode.continuousAutoFocus`.

This property's `CGPoint` value uses a coordinate system where `{0,0}` is the top-left of the picture area and `{1,1}` is the bottom-right. This coordinate system is always relative to a landscape device orientation with the home button on the right, regardless of the actual device orientation. You can convert between this coordinate system and view coordinates using `AVCaptureVideoPreviewLayer` methods.

This could be set to different locations every time (ex. focus to the PoI, where the user has indicated on screen via touch). Or it could just be set to one fixed location.

Configuring Capture Device (Focus)

AVCaptureDevice.FocusMode

Constants to specify the focus mode of a capture device.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
enum FocusMode : Int, @unchecked Sendable
```

Topics

Focus Modes

`case locked`

A mode that locks device focus.

`case autoFocus`

A mode that automatically adjusts the focus one time, and then locks focus.

`case continuousAutoFocus`

A mode that continuously monitors focus and autofocuses when necessary.

Configuring Capture Device (Focus)

isSmoothAutoFocusEnabled

A Boolean value that indicates whether smooth autofocus is in an enabled state on the device.

iOS 7.0+ | iPadOS 7.0+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var isSmoothAutoFocusEnabled: Bool { get set }
```

Discussion

On capable devices, you can enable a focusing mode in which the camera makes lens movements more slowly. This mode make focus transitions less visually intrusive, a behavior that you may want for video capture.

Before changing the value of this property, you must call `lockForConfiguration()` to acquire exclusive access to the device's configuration properties. Otherwise, setting the value of this property raises an exception. When you finish configuring the device, call `unlockForConfiguration()` to release the lock and allow other devices to configure the settings.

Configuring Capture Device (Exposure)

Managing the Exposure Mode

```
func isExposureModeSupported(AVCaptureDevice.ExposureMode) -> Bool
```

Returns a Boolean value that indicates whether a device supports the specified exposure mode.

```
var exposureMode: AVCaptureDevice.ExposureMode
```

The exposure mode for the device.

```
enum AVCaptureDevice.ExposureMode
```

Constants that specify the exposure mode of a capture device.

Setting an Exposure Point of Interest

```
var isExposurePointOfInterestSupported: Bool
```

A Boolean value that indicates whether the device supports a point of interest for exposure.

```
var exposurePointOfInterest: CGPoint
```

The point of interest for exposure.

Configuring Face-Driven Automatic Exposure

```
var isFaceDrivenAutoExposureEnabled: Bool
```

A Boolean value that indicates whether the device has face-driven autoexposure enabled.

```
var automaticallyAdjustsFaceDrivenAutoExposureEnabled: Bool
```

A Boolean value that indicates whether the device automatically adjusts face-driven autoexposure.

Configuring Exposure Manually

```
var exposureDuration: CMTime
```

The length of time over which exposure takes place.

```
var activeMaxExposureDuration: CMTime
```

The maximum exposure duration, in seconds, defined in the autoexposure algorithm.

```
var iso: Float
```

The current exposure ISO value.

```
var lensAperture: Float
```

The size of the lens diaphragm.

```
func setExposureModeCustom(duration: CMTime, iso: Float, completionHandler: ((CMTime) -
> Void)?)
```

Sets the exposure mode to a custom state, and locks exposure duration and ISO at explicit values.

Configuring Capture Device (Exposure)

exposurePointOfInterest

The point of interest for exposure.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var exposurePointOfInterest: CGPoint { get set }
```

Discussion

Setting a value for this property doesn't initiate an exposure rebalancing operation. To set exposure using a point of interest, first set this property's value, then set the `exposureMode` property to `AVCaptureDevice.ExposureMode.autoExpose` or `AVCaptureDevice.ExposureMode.continuousAutoExposure`.

This property's `CGPoint` value uses a coordinate system where `{0,0}` is the top-left of the picture area and `{1,1}` is the bottom-right. This coordinate system is always relative to a landscape device orientation with the home button on the right, regardless of the actual device orientation. You can convert between this coordinate system and view coordinates using `AVCaptureVideoPreviewLayer` methods.

Before changing the value of this property, you must call `lockForConfiguration()` to acquire exclusive access to the device's configuration properties. Otherwise, setting the value of this property raises an exception. When you're done configuring the device, call `unlockForConfiguration()` to release the lock and allow other devices to configure the settings.

This property is key-value observable.

Configuring Capture Device (Exposure)

exposureMode

The exposure mode for the device.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var exposureMode: AVCaptureDevice.ExposureMode { get set }
```

Discussion

Before changing the value of this property, you must call `lockForConfiguration()` to acquire exclusive access to the device's configuration properties. Otherwise, setting the value of this property raises an exception. When you're done configuring the device, call `unlockForConfiguration()` to release the lock and allow other devices to configure the settings.

This property is key-value observable.

AVCaptureDevice.ExposureMode

Constants that specify the exposure mode of a capture device.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
enum ExposureMode : Int, @unchecked Sendable
```

Topics

Exposure Modes

`case locked`

A mode that locks exposure for the device.

`case autoExpose`

A mode that automatically adjusts the exposure one time, and then locks exposure for the device.

`case continuousAutoExposure`

A mode that continuously monitors exposure levels and automatically adjusts exposure when necessary.

`case custom`

A mode where an app manually sets the exposure duration and ISO values.

Configuring Capture Device (Exposure)

isFaceDrivenAutoExposureEnabled

A Boolean value that indicates whether the device has face-driven autoexposure enabled.

iOS 15.4+ | iPadOS 15.4+ | Mac Catalyst 15.4+ | tvOS 17.0+

```
var isFaceDrivenAutoExposureEnabled: Bool { get set }
```

Discussion

Face-driven autoexposure takes a subject's face into account when performing automatic exposure adjustments. Enabling this feature can better expose subjects with darker skin tones or those who are backlit. For apps that link against iOS 15.4 or later, the value of this property defaults to `true` for devices that support autoexposure.

Before setting a value for this property, perform the following:

- Obtain exclusive access to the device by calling its `lockForConfiguration()` method.
- Set the value of the device's `automaticallyAdjustsFaceDrivenAutoExposureEnabled` property to `false`.

Attempting to set a value before performing these steps results in an exception.

When you finish configuring the device, unlock it by calling its `unlockForConfiguration()` method.

Important

Updating the state of this property doesn't initiate an exposure change. After setting a new value, set an appropriate `exposureMode` to apply the change.

automaticallyAdjustsFaceDrivenAutoExposureEnabled

A Boolean value that indicates whether the device automatically adjusts face-driven autoexposure.

iOS 15.4+ | iPadOS 15.4+ | Mac Catalyst 15.4+ | tvOS 17.0+

```
var automaticallyAdjustsFaceDrivenAutoExposureEnabled: Bool { get set }
```

Discussion

The value of this property defaults to `true` for devices that support autoexposure. If your app requires explicitly setting the state of `isFaceDrivenAutoExposureEnabled`, set this value to `false`.

To set this property value, you must call the device's `lockForConfiguration()` method to obtain exclusive access to configure it. Otherwise, attempting to set a value raises an exception. When you finish configuring the device, call `unlockForConfiguration()` to release the lock.

Configuring Capture Device (Exposure)

activeMaxExposureDuration

The maximum exposure duration, in seconds, defined in the autoexposure algorithm.

iOS 12.0+ | iPadOS 12.0+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var activeMaxExposureDuration: CMTime { get set }
```

Discussion

When you set the exposureMode to `AVCaptureDevice.ExposureMode.autoExpose` or `AVCaptureDevice.ExposureMode.continuousAutoExposure`, the autoexposure algorithm picks a default maximum exposure duration that's tuned for the current configuration, balancing low light image quality with motion preservation. By querying or key-value observing this property, you can determine the current maximum exposure duration in use.

You may also override the default value by setting this property to a value between the format's `minExposureDuration` and `maxExposureDuration` values. The system throws an exception if you pass an out-of-bounds exposure value.

Setting the property to the special value of `invalid` resets the autoexposure maximum duration to the device's default for your current configuration. When the device's `activeFormat` or the capture session's `sessionPreset` changes, this property resets to the default max exposure duration for the new format or session preset.

On some devices, the auto exposure algorithm picks a different maximum exposure duration for a given format depending on whether you used the `sessionPreset` or `activeFormat` APIs to set to set the format. To ensure uniform default handling of maximum exposure duration, set the value of a capture input's `unifiedAutoExposureDefaultsEnabled` property to `true`.

Configuring Capture Device (White Balance)

Configuring Automatic White Balance

```
func isWhiteBalanceModeSupported(AVCaptureDevice.WhiteBalanceMode) -> Bool
```

Returns a Boolean value that indicates whether the device supports the specified white balance mode.

```
var whiteBalanceMode: AVCaptureDevice.WhiteBalanceMode
```

The current white balance mode.

```
enum AVCaptureDevice.WhiteBalanceMode
```

Constants to specify the white balance mode of a capture device.

Configuring Capture Device (White Balance)

whiteBalanceMode

The current white balance mode.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var whiteBalanceMode: AVCaptureDevice.WhiteBalanceMode { get set }
```

Discussion

Before changing the value of this property, you must call `lockForConfiguration()` to acquire exclusive access to the device's configuration properties. Otherwise, setting the value of this property raises an exception. When you're done configuring the device, call `unlockForConfiguration()` to release the lock and allow other devices to configure the settings.

This property is key-value observable.

AVCaptureDevice.WhiteBalanceMode

Constants to specify the white balance mode of a capture device.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
enum WhiteBalanceMode : Int, @unchecked Sendable
```

Topics

White Balance Modes

`case locked`

A mode that locks the white balance state.

`case autoWhiteBalance`

A mode that automatically manages white balance.

`case continuousAutoWhiteBalance`

A mode that continuously monitors white balance and adjusts when necessary.

Configuring Capture Device (Zoom)

videoZoomFactor

A value that controls the cropping and enlargement of images captured by the device.

iOS 7.0+ | iPadOS 7.0+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
var videoZoomFactor: CGFloat { get set }
```

Discussion

This value is a multiplier. For example, a value of `2.0` doubles the size of an image's subject (and halves the field of view). Allowed values range from `1.0` (full field of view) to the value of the active format's `videoMaxZoomFactor` property. Setting the value of this property jumps immediately to the new zoom factor. For a smooth transition, use the `ramp(toVideoZoomFactor:withRate:)` method.

The device achieves a zoom effect by cropping around the center of the image captured by the sensor. At low zoom factors, the cropped images is equal to or larger than the output size. At higher zoom factors, the device must scale the cropped image up to the output size, resulting in a loss of image quality. The active format's `videoZoomFactorUpscaleThreshold` property indicates the factors at which upscaling occurs.

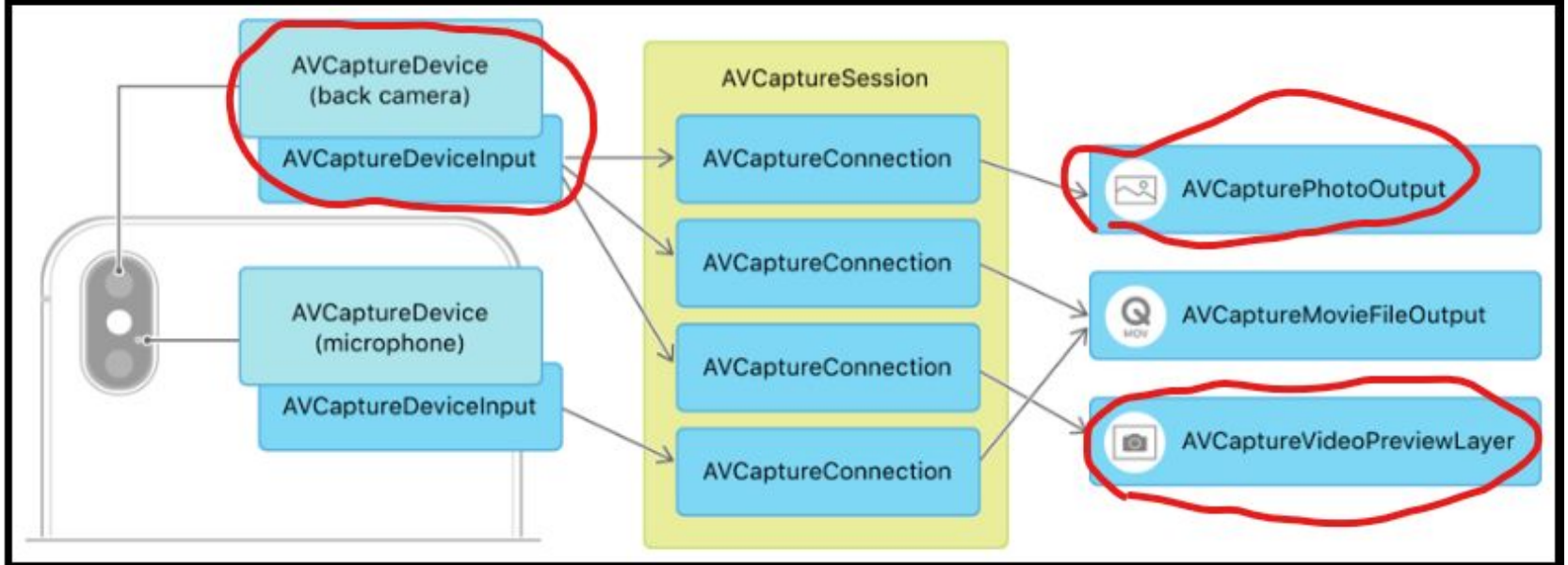
Before changing the value of this property, you must call `lockForConfiguration()` to acquire exclusive access to the device's configuration properties. Otherwise, setting the value of this property raises an exception. When you finish configuring the device, call `unlockForConfiguration()` to release the lock and allow other devices to configure the settings.

Capture Session Setup

Overview

An `AVCaptureSession` is the basis for all media capture in iOS and macOS. It manages your app's exclusive access to the OS capture infrastructure and capture devices, as well as the flow of data from input devices to media outputs. How you configure connections between inputs and outputs defines the capabilities of your capture session. For example, the diagram below shows a capture session that can capture both photos and movies and provides a camera preview, using the iPhone back camera and microphone.

Figure 1 Architecture of an example capture session



Set Up Device Input for Capture Session

AVCaptureDeviceInput

An object that provides media input from a capture device to a capture session.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+ | visionOS 1.0+

```
class AVCaptureDeviceInput : AVCaptureInput
```

Overview

This class is a concrete subclass of `AVCaptureInput` that you use to connect a capture device to a capture session.

Topics

Creating an Input

`init(device: AVCaptureDevice)`

Creates an input for the specified capture device.

Configuring Input Properties

`var unifiedAutoExposureDefaultsEnabled: Bool`

A Boolean value that indicates whether the input enables unified auto-exposure defaults.

`var videoMinFrameDurationOverride: CMTime`

A time value that acts as a modifier to a capture device's active video minimum frame duration.

Set Up Photo Output for Capture Session

AVCapturePhotoOutput

A capture output for still image, Live Photos, and other photography workflows.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class AVCapturePhotoOutput : AVCaptureOutput
```

Overview

`AVCapturePhotoOutput` provides an interface for capture workflows related to still photography. In addition to basic capture of still images, a photo output supports RAW-format capture, bracketed capture of multiple images, Live Photos, and wide-gamut color. You can output captured photos in a variety of formats and codecs, including RAW format DNG files, HEVC format HEIF files, and JPEG files.

To capture photos with the `AVCapturePhotoOutput` class, follow these steps:

1. Create an `AVCapturePhotoOutput` object. Use its properties to determine supported capture settings and to enable certain features (for example, whether to capture Live Photos).
2. Create and configure an `AVCapturePhotoSettings` object to choose features and settings for a specific capture (for example, whether to enable image stabilization or flash).
3. Capture an image by passing your photo settings object to the `capturePhoto(with:delegate:)` method along with a delegate object implementing the `AVCapturePhotoCaptureDelegate` protocol. The photo capture output then calls your delegate to notify you of significant events during the capture process.

Some photo capture settings, such as the `flashMode` property, include options for automatic behavior. For such settings, the photo output determines whether to use that feature at the moment of capture—you don't know when requesting a capture whether the feature will be enabled when the capture completes. When the photo capture output calls your `AVCapturePhotoCaptureDelegate` methods with information about the completed or in-progress capture, it also provides an `AVCaptureResolvedPhotoSettings` object that details which automatic features are set for that capture. The resolved settings object's `uniqueID` property matches the `uniqueID` value of the `AVCapturePhotoSettings` object you used to request capture.

Enabling certain photo features (Live Photo capture and high resolution capture) requires a reconfiguration of the capture render pipeline. To opt into these features, set the `isHighResolutionCaptureEnabled`, `isLivePhotoCaptureEnabled`, and `isLivePhotoAutoTrimmingEnabled` properties before calling your `AVCaptureSession` object's `startRunning()` method. Changing any of these properties while the session is running disrupts the capture render pipeline: Live Photo captures in progress end immediately, unfulfilled photo requests abort, and video preview temporarily freezes.

Using a photo capture output adds other requirements to your `AVCaptureSession` object:

- A capture session can't support both Live Photo capture and movie file output. If your capture session includes an `AVCaptureMovieFileOutput` object, the `isLivePhotoCaptureSupported` property becomes `false`. (As an alternative, you can use the `AVCaptureVideoDataOutput` class to output video buffers at the same resolution as a simultaneous Live Photo capture).
- A capture session can't contain both an `AVCapturePhotoOutput` object and an `AVCaptureStillImageOutput` object. The `AVCapturePhotoOutput` class includes all functionality of (and deprecates) the `AVCaptureStillImageOutput` class.

The `AVCapturePhotoOutput` class implicitly supports wide-gamut color photography. If the source `AVCaptureDevice` object's `activeColorSpace` value is `AVCaptureColorSpace.P3_D65`, the capture output produces photos with wide color information (unless your `AVCapturePhotoSettings` object specifies an output format that doesn't support wide color).

Set Up Photo Output for Capture Session

Managing Responsive Capture

`var captureReadiness: AVCapturePhotoOutput.CaptureReadiness`

A value that specifies whether the photo output is ready to respond to new capture requests in a timely manner.

`enum AVCapturePhotoOutput.CaptureReadiness`

Constants that indicate whether the output is ready to receive capture requests.

`var isAutoDeferredPhotoDeliveryEnabled: Bool`

A Boolean value that indicates the enabled state of automatic deferred photo delivery.

`var isAutoDeferredPhotoDeliverySupported: Bool`

A Boolean value that indicates whether the photo output supports deferred photo delivery.

`var isFastCapturePrioritizationSupported: Bool`

A Boolean value that indicates whether the photo output supports fast capture prioritization.

`var isFastCapturePrioritizationEnabled: Bool`

A Boolean value that indicates whether the output enables fast capture prioritization.

`var isResponsiveCaptureSupported: Bool`

A Boolean value that indicates whether the photo output supports responsive capture.

`var isResponsiveCaptureEnabled: Bool`

A Boolean value that indicates whether the photo output configuration enables responsive capture.

`var isZeroShutterLagSupported: Bool`

A Boolean value that indicates whether the photo output supports zero shutter lag.

`var isZeroShutterLagEnabled: Bool`

A Boolean value that indicates whether the photo output configuration enables zero shutter lag.

Setting the Capture Prioritization

`var maxPhotoQualityPrioritization: AVCapturePhotoOutput.QualityPrioritization`

The highest quality the photo output should prepare to deliver on a capture-by-capture basis.

Configuring Live Photo Capture

`var isLivePhotoCaptureSupported: Bool`

A Boolean value that indicates whether the capture output currently supports Live Photo capture.

`var isLivePhotoCaptureEnabled: Bool`

A Boolean value that indicates whether to configure the capture pipeline for Live Photo capture.

Configuring Depth Data Capture

`var isDepthDataDeliverySupported: Bool`

A Boolean value indicating whether the capture output currently supports depth data capture.

`var isDepthDataDeliveryEnabled: Bool`

A Boolean value that specifies whether to configure the capture pipeline for depth data capture.

Configuring Portrait Effects Matte Capture

`var isPortraitEffectsMatteDeliveryEnabled: Bool`

A Boolean value indicating whether the capture output generates a portrait effects matte.

`var isPortraitEffectsMatteDeliverySupported: Bool`

A Boolean value indicating whether the capture output currently supports delivery of a portrait effects matte.

`var portraitEffectsMatte: AVPortraitEffectsMatte?`

The portrait effects matte captured with the photo.

Set Up Camera Preview Output for Capture Session

AVCaptureVideoPreviewLayer

A Core Animation layer that displays video from a camera device.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class AVCaptureVideoPreviewLayer : CALayer
```

Overview

Use this layer to provide a preview of the content the camera captures. A convenient way to use this class in iOS is to set it as the backing layer for a view as shown below.

```
class PreviewView: UIView {

    // Use AVCaptureVideoPreviewLayer as the view's backing layer.
    override class var layerClass: AnyClass {
        AVCaptureVideoPreviewLayer.self
    }

    var previewLayer: AVCaptureVideoPreviewLayer {
        layer as! AVCaptureVideoPreviewLayer
    }

    // Connect the layer to a capture session.
    var session: AVCaptureSession? {
        get { previewLayer.session }
        set { previewLayer.session = newValue }
    }
}
```

```
/// This view allows the app to display an `AVCapturePreviewLayer` in SwiftUI.
/// It's used by `CameraView` to interact with the `CameraModel`.
struct CameraPreviewView: UIViewRepresentable {
    let previewViewCornerRadius: CGFloat = 50
    //let session: AVCaptureSession
    @ObservedObject var model: CameraViewModel

    class PreviewView: UIView {

        var videoPreviewLayer: AVCaptureVideoPreviewLayer {
            guard let layer = layer as? AVCaptureVideoPreviewLayer else {
                fatalError("Expected `AVCaptureVideoPreviewLayer` type for layer. Check
                    PreviewView.layerClass implementation.")
            }
            return layer
        }

        var session: AVCaptureSession? {
            get {
                return videoPreviewLayer.session
            }
            set {
                videoPreviewLayer.session = newValue
            }
        }

        override class var layerClass: AnyClass {
            return AVCaptureVideoPreviewLayer.self
        }
    }

    func makeUIView(context: Context) -> PreviewView {
        let view = PreviewView()
        view.videoPreviewLayer.session = model.session

        // Set the view's initial state.
        view.backgroundColor = .black
        view.videoPreviewLayer.cornerRadius = 0
        view.videoPreviewLayer.connection?.videoOrientation = .portrait

        model.rootLayer = view.layer

        return view
    }

    func updateUIView(_ uiView: PreviewView, context: Context) {  }
```

Configure Capture Session

Connect Inputs and Outputs to the Session

All capture sessions need at least one capture input and capture output. Capture inputs (`AVCaptureInput` subclasses) are media sources—typically recording devices like the cameras and microphone built into an iOS device or Mac. Capture outputs (`AVCaptureOutput` subclasses) use data provided by capture inputs to produce media, like image and movie files.

To use a camera for video input (to capture photos or movies), select an appropriate `AVCaptureDevice`, create a corresponding `AVCaptureDeviceInput`, and add it to the session:

```
captureSession.beginConfiguration()
let videoDevice = AVCaptureDevice.default(.builtInWideAngleCamera,
                                          for: .video, position: .unspecified)
guard
    let videoDeviceInput = try? AVCaptureDeviceInput(device: videoDevice!),
    captureSession.canAddInput(videoDeviceInput)
    else { return }
captureSession.addInput(videoDeviceInput)
```

> **Note**
> iOS offers several other ways to select a camera device. For more information, see Choosing a Capture Device.

Next, add outputs for the kinds of media you plan to capture from the camera you've selected. For example, to enable capturing photos, add an `AVCapturePhotoOutput` to the session:

```
let photoOutput = AVCapturePhotoOutput()
guard captureSession.canAddOutput(photoOutput) else { return }
captureSession.sessionPreset = .photo
captureSession.addOutput(photoOutput)
captureSession.commitConfiguration()
```

AVCaptureSession

An object that configures capture behavior and coordinates the flow of data from input devices to capture outputs.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+ | visionOS 1.0+

```
class AVCaptureSession : NSObject
```

> **Important**
> Call `beginConfiguration()` before changing a session's inputs or outputs, and call `commitConfiguration()` after making changes.

Call the `startRunning()` method to start the flow of data from the inputs to the outputs, and call the `stopRunning()` method to stop the flow.

> **Important**
> The `startRunning()` method is a blocking call which can take some time, therefore start the session on a serial dispatch queue so that you don't block the main queue (which keeps the UI responsive). See AVCam: Building a Camera App for an implementation example.

You use the `sessionPreset` property to customize the quality level, bitrate, or other settings for the output. Most common capture configurations are available through session presets; however, some specialized options (such as high frame rate) require directly setting a capture format on an `AVCaptureDevice` instance.

Configure Capture Session

AVCaptureSession.Preset

Presets that define standard configurations for a capture session.

iOS 4.0+ | iPadOS 4.0+ | macOS 10.7+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
struct Preset
```

Overview

Setting a `sessionPreset` value provides a convenient way to configure a capture session for common use cases.

Topics

Quality Levels

`static let high: AVCaptureSession.Preset`

A preset suitable for capturing high-quality output.

`static let medium: AVCaptureSession.Preset`

A preset suitable for capturing medium-quality output.

`static let low: AVCaptureSession.Preset`

A preset suitable for capturing low-quality output.

Photo

`static let photo: AVCaptureSession.Preset`

A preset suitable for capturing high-resolution photo quality output.

Need to set the preset to .photo in order to get 3024 x 4032 photos.
Otherwise, the saved image comes out as 1080 x 1920

```
    func configureSession() {
        guard self.session.canAddInput(inputCamera), self.session.canAddOutput(photoCaptureManager.photoOutput)
        else { fatalError("Session could not be configured") }

        self.session.beginConfiguration()


        self.session.addInput(inputCamera)
        self.session.addOutput(photoCaptureManager.photoOutput)
        self.session.setPhotoPreset()

        self.session.commitConfiguration()
    }

    func startRunning() async {
        if !session.isRunning {
            self.configureSession()
            session.startRunning()
            await MainActor.run { setPreviewSession() }
        }
    }

    func stopRunning() { if session.isRunning { session.stopRunning() } }
}

// MARK: AVCaptureSession Photo Preset
extension AVCaptureSession {
    func setPhotoPreset() { self.sessionPreset = .photo }
}

// MARK: Video Preview Output
extension CaptureSession {
    /// This needs to run on Main thread since this updates the UIView
    @MainActor
    func setPreviewSession() { self.outputCameraPreview.session = self.session }
}
```

Capturing Photos Setup

Setting videoRotationAngle before capturePhoto

Capture a Photo

Taking a photo happens on the session queue. The process begins by updating the `AVCapturePhotoOutput` connection to match the video orientation of the video preview layer. This enables the camera to accurately capture what the user sees onscreen:

```
if let photoOutputConnection = self.photoOutput.connection(with: .video) {
    photoOutputConnection.videoRotationAngle = videoRotationAngle
}
```

After aligning the outputs, AVCam proceeds to create `AVCapturePhotoSettings` to configure capture parameters such as focus, flash, and resolution:

```
var photoSettings = AVCapturePhotoSettings()

// Capture HEIF photos when supported.
if self.photoOutput.availablePhotoCodecTypes.contains(AVVideoCodecType.hevc) {
    photoSettings = AVCapturePhotoSettings(format: [AVVideoCodecKey: AVVideoCodecType.hevc])
} else {
    photoSettings = AVCapturePhotoSettings()
}

// Set the flash to auto mode.
if self.videoDeviceInput.device.isFlashAvailable {
    photoSettings.flashMode = .auto
}

// Enable high-resolution photos.
photoSettings.maxPhotoDimensions = self.photoOutput.maxPhotoDimensions
if !photoSettings.availablePreviewPhotoPixelFormatTypes.isEmpty {
    photoSettings.previewPhotoFormat = [kCVPixelBufferPixelFormatTypeKey as String: photoSettings.__av
}
if self.livePhotoMode == .on && self.photoOutput.isLivePhotoCaptureSupported { // Live Photo Capture i
    photoSettings.livePhotoMovieFileURL = livePhotoMovieUniqueTemporaryDirectoryFileURL()
}
photoSettings.photoQualityPrioritization = self.photoQualityPrioritizationMode
```

Do I need to set this before capturing photo??

Yes. This needs to be set so that what we see on screen is what we get when we take photos or do post processing on the video

videoRotationAngle

A rotation angle the connection applies to a video flowing through it.

iOS 17.0+ | iPadOS 17.0+ | macOS 14.0+ | Mac Catalyst 17.0+ | tvOS 17.0+

```
var videoRotationAngle: CGFloat { get set }
```

Discussion

Your app can set a video rotation angle that it gets from an `AVCaptureDevice.RotationCoordinator` instance's `videoRotationAngleForHorizonLevelCapture` or `videoRotationAngleForHorizonLevelPreview` property. The rotation angle only applies to video or depth connections, similar to `isVideoMirrored`, and can be any angle that `isVideoRotationAngleSupported(_:)` returns `true` for.

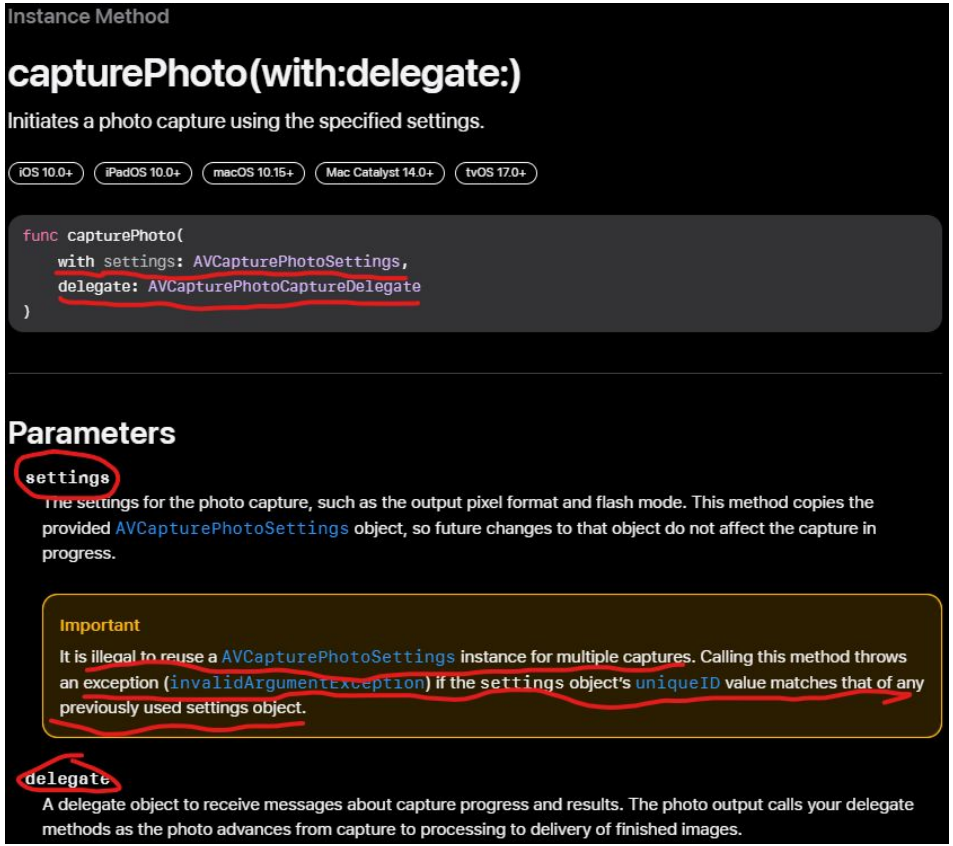
Not all capture connections rotate each frame. For example, a video connection to an `AVCaptureMovieFileOutput` or `AVCapturePhotoOutput` instance applies a rotation with a QuickTime track matrix or with EXIF tags, respectively.

Capture connections to `AVCaptureVideoDataOutput` and `AVCaptureDepthDataOutput` instances rotate video frames they provide to their `captureOutput(_:didOutput:from:)` and `depthDataOutput(_:didOutput:timestamp:connection:)` delegate methods, respectively. Each `AVCaptureVideoDataOutput` instance uses hardware acceleration to rotate every frame.

> **Tip**
> Avoid potential performance issues by only rotating video with a capture connection when necessary.

You can rotate the video of a movie file you record with an `AVAssetWriter` instance by applying the rotation to an `AVAssetWriterInput` instance's `transform` property. This approach avoids the performance costs that come with rotating each video frame.

Using capturePhoto method in AVCapturePhotoOutput



Instance Method

capturePhoto(with:delegate:)

Initiates a photo capture using the specified settings.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
func capturePhoto(
    with settings: AVCapturePhotoSettings,
    delegate: AVCapturePhotoCaptureDelegate
)
```

Parameters

settings

The settings for the photo capture, such as the output pixel format and flash mode. This method copies the provided `AVCapturePhotoSettings` object, so future changes to that object do not affect the capture in progress.

> **Important**
>
> It is illegal to reuse a `AVCapturePhotoSettings` instance for multiple captures. Calling this method throws an exception (`invalidArgumentException`) if the `settings` object's `uniqueID` value matches that of any previously used settings object.

delegate

A delegate object to receive messages about capture progress and results. The photo output calls your delegate methods as the photo advances from capture to processing to delivery of finished images.

Using capturePhoto (AVCapturePhotoSettings)

AVCapturePhotoSettings

A specification of the features and settings to use for a single photo capture request.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class AVCapturePhotoSettings : NSObject
```

Overview

To take a photo, you create and configure a `AVCapturePhotoSettings` object, then pass it to the `AVCapturePhotoOutput capturePhoto(with:delegate:)` method.

A `AVCapturePhotoSettings` instance can include any combination of settings, regardless of whether that combination is valid for a given capture session. When you initiate a capture by passing a photo settings object to the `AVCapturePhotoOutputcapturePhoto(with:delegate:)` method, the photo capture output validates your settings to ensure deterministic behavior. For example, the `flashMode` setting must specify a value that is present in the photo output's `supportedFlashModes` array. For detailed validation rules, see each property description below.

Important

It is illegal to reuse a `AVCapturePhotoSettings` instance for multiple captures. Calling the `capturePhoto(with:delegate:)` method throws an exception (`invalidArgumentException`) if the `settings` object's `uniqueID` value matches that of any previously used settings object.

To reuse a specific combination of settings, use the `init(from:)` initializer to create a new, unique `AVCapturePhotoSettings` instance from an existing photo settings object.

Configuring Photo Settings

`var flashMode: AVCaptureDevice.FlashMode`

A setting for whether to fire the flash when capturing photos.

`var isAutoRedEyeReductionEnabled: Bool`

A Boolean value that determines whether to use auto red-eye reduction on flash captures.

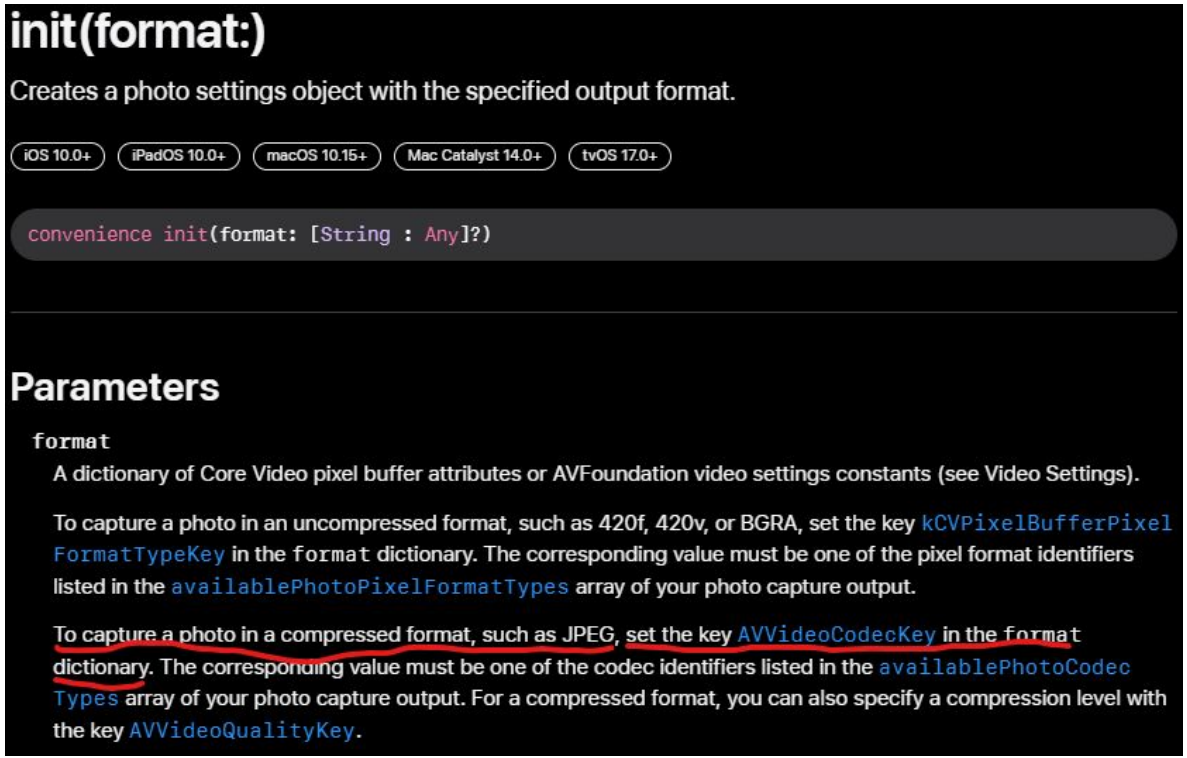
`var maxPhotoDimensions: CMVideoDimensions`

The maximum resolution of the photo to capture.

`var photoQualityPrioritization: AVCapturePhotoOutput.QualityPrioritization`

A setting that indicates how to prioritize photo quality against speed of photo delivery.

Using capturePhoto (AVCapturePhotoSettings)



init(format:)

Creates a photo settings object with the specified output format.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
convenience init(format: [String : Any]?)
```

Parameters

`format`

A dictionary of Core Video pixel buffer attributes or AVFoundation video settings constants (see Video Settings).

To capture a photo in an uncompressed format, such as 420f, 420v, or BGRA, set the key `kCVPixelBufferPixelFormatTypeKey` in the `format` dictionary. The corresponding value must be one of the pixel format identifiers listed in the `availablePhotoPixelFormatTypes` array of your photo capture output.

To capture a photo in a compressed format, such as JPEG, set the key `AVVideoCodecKey` in the `format` dictionary. The corresponding value must be one of the codec identifiers listed in the `availablePhotoCodecTypes` array of your photo capture output. For a compressed format, you can also specify a compression level with the key `AVVideoQualityKey`.

In our case, use AVCapturePhotoSettings(format: [AVCideoCodecKey: AVCideoCodecType.jpeg])

Using capturePhoto (AVCapturePhotoCaptureDelegate)

AVCapturePhotoCaptureDelegate

Methods for monitoring progress and receiving results from a photo capture output.

iOS 10.0+ | iPadOS 10.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
protocol AVCapturePhotoCaptureDelegate
```

Overview

You implement methods in the `AVCapturePhotoCaptureDelegate` protocol to be notified of progress and results when capturing photos with the `AVCapturePhotoOutput` class.

To capture a photo, you pass an object implementing this protocol to the `capturePhoto(with:delegate:)` method, along with a settings object that describes the capture to be performed. As the capture proceeds, the photo output calls several of the methods in this protocol on your delegate object, providing information about the capture's progress and delivering the resulting photos.

Which delegate methods the photo output calls depends on the photo settings you initiate capture with. All methods in this protocol are optional at compile time, but at run time your delegate object must respond to certain methods depending on your photo settings:

- If you request a still photo capture (by specifying image formats or file types), your delegate either must implement the `photoOutput(_:didFinishProcessingPhoto:error:)` method, or must implement methods listed in `Receiving Capture Results (Deprecated)` corresponding to whether you request capture in RAW format, processed format, or both.
- If you request Live Photo capture (by setting the `livePhotoMovieFileURL` property to a non-`nil` value), your delegate must implement the `photoOutput(_:didFinishProcessingLivePhotoToMovieFileAt:duration:photoDisplayTime:resolvedSettings:error:)` method.

The capture output validates these requirements when you call the `capturePhoto(with:delegate:)` method. If your delegate does not meet these requirements, that method raises an exception.

Topics

Monitoring Capture Progress

`func photoOutput(AVCapturePhotoOutput, willBeginCaptureFor: AVCaptureResolvedPhotoSettings)`

Notifies the delegate that the capture output has resolved settings and will soon begin its capture process.

`func photoOutput(AVCapturePhotoOutput, willCapturePhotoFor: AVCaptureResolvedPhotoSettings)`

Notifies the delegate that photo capture is about to occur.

`func photoOutput(AVCapturePhotoOutput, didCapturePhotoFor: AVCaptureResolvedPhotoSettings)`

Notifies the delegate that the photo has been taken.

`func photoOutput(AVCapturePhotoOutput, didFinishCaptureFor: AVCaptureResolvedPhotoSettings, error: Error?)`

Notifies the delegate that the capture process is complete.

Receiving Capture Results

`func photoOutput(AVCapturePhotoOutput, didFinishProcessingPhoto: AVCapturePhoto, error: Error?)`

Provides the delegate with the captured image and associated metadata resulting from a photo capture.

`func photoOutput(AVCapturePhotoOutput, didFinishRecordingLivePhotoMovieForEventualFileAt: URL, resolvedSettings: AVCaptureResolvedPhotoSettings)`

Notifies the delegate that the movie content of a Live Photo has finished recording.

`func photoOutput(AVCapturePhotoOutput, didFinishProcessingLivePhotoToMovieFileAt: URL, duration: CMTime, photoDisplayTime: CMTime, resolvedSettings: AVCaptureResolvedPhotoSettings, error: Error?)`

Provides the delegate the movie file URL resulting from a Live Photo capture.

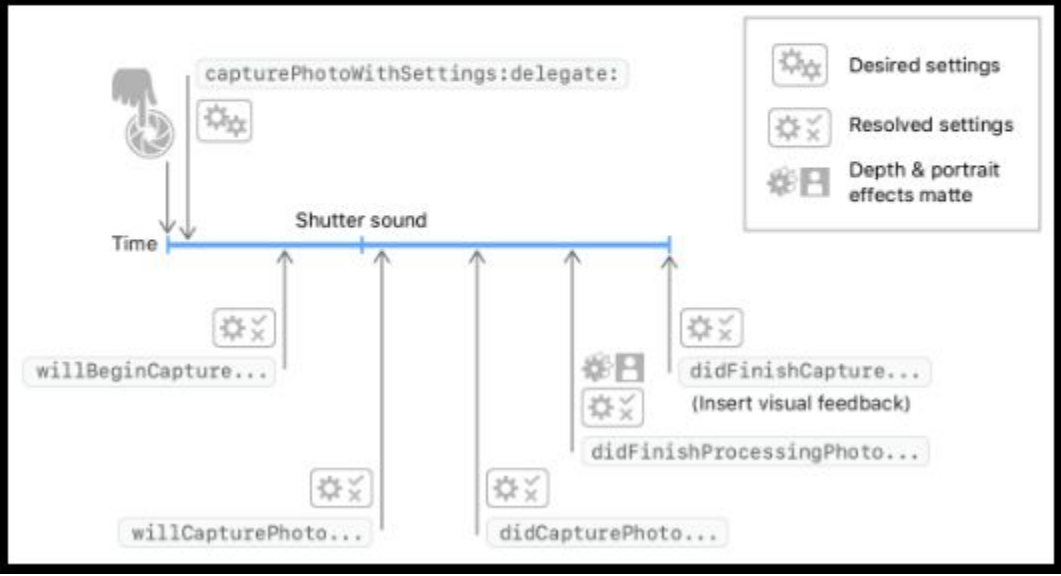
`func photoOutput(AVCapturePhotoOutput, didFinishCapturingDeferredPhotoProxy: AVCaptureDeferredPhotoProxy?, error: Error?)`

Tells the delegate when the system finishes capturing the photo proxy.

Using capturePhoto (AVCapturePhotoCaptureDelegate)

Track Results Through a Photo Capture Delegate

The method `capturePhoto` only begins the process of taking a photo. The rest of the process happens in delegate methods that the app implements.



Using capturePhoto (AVCapturePhotoCaptureDelegate)

The code below shows one way to manage multiple photo capture delegate objects:

```
class PhotoCaptureProcessor: NSObject, AVCapturePhotoCaptureDelegate {
    var completionHandler: () -> () = {}
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishCaptureFor resolvedSettings: AVCaptureRe
        completionHandler()
    }
    // ... other delegate methods to handle capture results...
}

// Keep a set of in-progress capture delegates.
var capturesInProgress = Set<PhotoCaptureProcessor>()

func shootPhoto() {
    // Make a new capture delegate for each capture and add it to the set.
    let captureProcessor = PhotoCaptureProcessor()
    capturesInProgress.insert(captureProcessor)

    // Schedule for the capture delegate to be removed from the set after capture.
    captureProcessor.completionHandler = { [weak self] in
        self?.capturesInProgress.remove(captureProcessor); return
    }

    self.photoOutput.capturePhoto(with: self.settingsForNextPhoto(), delegate: captureProcessor)
}
```

In this Apple's example code, they keep photo capture delegates in a set and remove them once they have been used to process associated photos.

I think this method uses a single thread (ie. DispatchQueue specifically configured for this process) to call on capturePhoto method.

In order to eliminate the necessity of using a set to keep the photo capture delegates in queue, I tried using a separate Task for each call to capturePhoto, as will be shown in the next slides.

**This alone did not solve the problem. Had to implement the Set data structure in order to keep reference to the photoCaptureDelegate. But still can use Task to call separate threads on top of this.

Using capturePhoto (AVCapturePhotoCaptureDelegate)

```swift
// MARK: Capture & Save Photo
extension AVCapturePhotoOutput {
    func saveFrame(to captureDir: String) {
        // Creates threads for saving photo when needed
        Task(priority: .background) {
            // Setup delegate & AVcapturePhotoSettings
            let photoCaptureSettings = AVCapturePhotoSettings.createConfiguredAVCapturePhotoSettings()
            let photoCaptureDelegate = PhotoCaptureProcessor(with: { photo in
                // Save Photo to a Document folder with URL of captureDir
                // MARK: Should be filled in

            })
            // Capture Photo
            self.capturePhoto(with: photoCaptureSettings, delegate: photoCaptureDelegate)
        }
    }
}
```

Using capturePhoto (AVCapturePhotoCaptureDelegate)

```swift
import Foundation
import AVFoundation

class PhotoCaptureProcessor: NSObject, AVCapturePhotoCaptureDelegate {

    private var photoProcessingHandler: (_ photo: AVCapturePhoto) -> Void

    init(with photoProcessingHandler: @escaping (_ photo: AVCapturePhoto) -> Void) {
        self.photoProcessingHandler = photoProcessingHandler
    }

    // MARK: Monitoring Capture Progress
    func photoOutput(_ output: AVCapturePhotoOutput, willBeginCaptureFor resolvedSettings: AVCaptureResolvedPhotoSettings) {
        // I think this is where the shutter sound should be killed
        // since shutter sound happens between willBeginCaptureFor and willCapturePhotoFor
        AudioServicesDisposeSystemSoundID(1108)
    }
    func photoOutput(_ output: AVCapturePhotoOutput, willCapturePhotoFor resolvedSettings: AVCaptureResolvedPhotoSettings) {  }
    func photoOutput(_ output: AVCapturePhotoOutput, didCapturePhotoFor resolvedSettings: AVCaptureResolvedPhotoSettings)  {  }
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishCaptureFor resolvedSettings: AVCaptureResolvedPhotoSettings, error: Error?) {  }

    // MARK: Receiving Capture Results
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVCapturePhoto, error: Error?) {
        // Where captured photo should be caught and saved
        photoProcessingHandler(photo)
    }

    // MARK: Receiving Capture Results for Live Photo
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishRecordingLivePhotoMovieForEventualFileAt outputFileURL: URL, resolvedSettings: AVCaptureResolvedPhotoSettings) {  }
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingLivePhotoToMovieFileAt outputFileURL: URL, duration: CMTime, photoDisplayTime: CMTime, resolvedSettings:
        AVCaptureResolvedPhotoSettings, error: Error?) {  }
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishCapturingDeferredPhotoProxy deferredPhotoProxy: AVCaptureDeferredPhotoProxy?, error: Error?) {  }
}
```

Saving Photos
(To Photo Library - Not necessary for our case)

Configure Info.plist

Overview

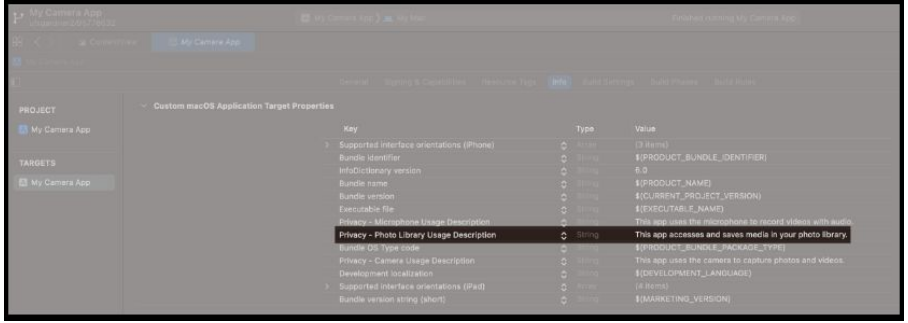
When you complete a photo capture with `AVCapturePhotoOutput`, you receive an `AVCapturePhoto` object that contains the image data, camera metadata, and any auxiliary images you requested, such as thumbnails or depth maps. You can retrieve this data individually from the `AVCapturePhoto` object. Or you can call its `fileDataRepresentation()` method to get a `Data` object that's ready to save using the codec and file format you requested for that photo in `AVCapturePhotoSettings`.

After capturing a photo, use PhotoKit to add that data to the user's photo library.

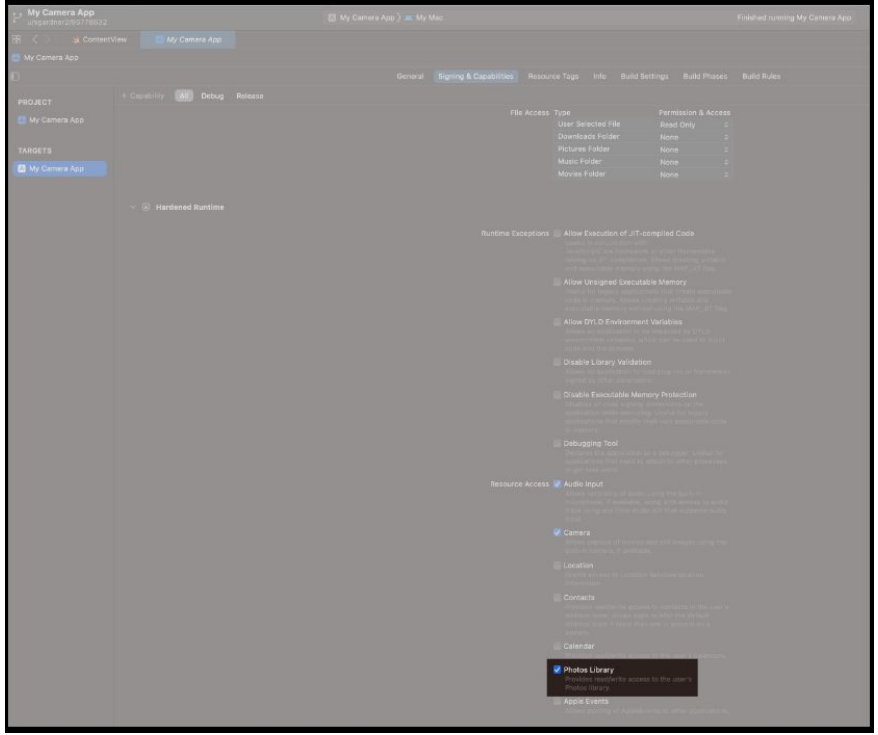
> **Note**
>
> If your app only needs to access content in the photo library, you can use the PhotoKit Photos picker instead, which doesn't require you to request access from the user. To learn more, see Selecting Photos and Videos in iOS.

Configure properties and capabilities for your app targets

You can use PhotoKit to enable read/write access to the user's photo library. To do so, provide a static message for the `NSPhotoLibraryUsageDescription` key in the `Info.plist` file to display to the user when your app requests access.



In macOS, you also need to enable the `Photos Library Entitlement` in the Signing & Capabilities tab for your app targets.



Important

Your app needs to contain the appropriate key in its `Info.plist` file, and the appropriate entitlement enabled in macOS, before it requests authorization or attempts to use a device. Otherwise, the system terminates your app.

Request permission to access the user's photo library

Use the `PHPhotoLibrary requestAuthorization(for:handler:)` to request access to the photo library at an appropriate time, such as when the user first opens your app's camera feature. Here's an example:

```
var isPhotoLibraryReadWriteAccessGranted: Bool {
    get async {
        let status = PHPhotoLibrary.authorizationStatus(for: .readWrite)

        // Determine if the user previously authorized read/write access.
        var isAuthorized = status == .authorized

        // If the system hasn't determined the user's authorization status,
        // explicitly prompt them for approval.
        if status == .notDetermined {
            isAuthorized = await PHPhotoLibrary.requestAuthorization(for: .readWrite) == .authorized
        }

        return isAuthorized
    }
}
```

Note

Don't wait to request access to the photo library until after the user takes their first photo because the permission alert prevents their ability to take multiple photos.

Use a Creation Request to add a photo asset

Perform these steps to receive a captured photo and save it to the photo library:

1. Adopt the `AVCapturePhotoCaptureDelegate` protocol and implement its `photoOutput(_:didFinishProcessingPhoto:error:)` method to receive a callback for each photo delivered in a capture request.
2. Call `fileDataRepresentation()` on the `AVCapturePhoto` object provided by the protocol method to receive a data object containing the photo image data and its attachments, such as camera metadata and auxilliary images.
3. Create a `PHAssetCreationRequest` to add the photo resource.

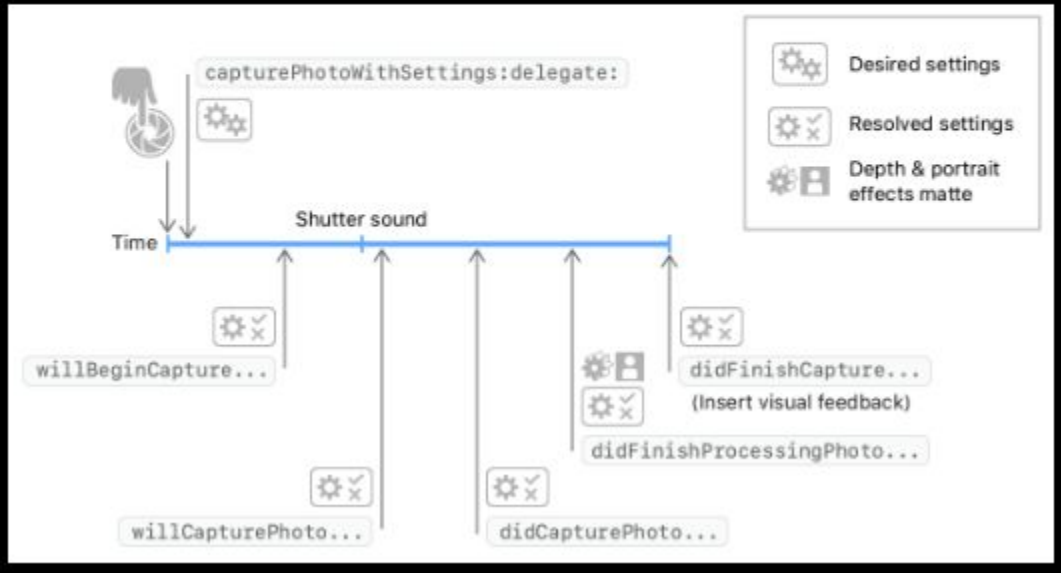
The following code provides an example of this workflow:

```
func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVCapturePhoto, error
    if let error {
        print("Error processing photo: \(error.localizedDescription)")
        return
    }

    Task {
        await save(photo: photo)
    }
}

func save(photo: AVCapturePhoto) async {
    // Confirm the user granted read/write access.
    guard await isPhotoLibraryReadWriteAccessGranted else { return }

    // Create a data representation of the photo and its attachments.
    if let photoData = photo.fileDataRepresentation() {
        PHPhotoLibrary.shared().performChanges {
            // Save the photo data.
            let creationRequest = PHAssetCreationRequest.forAsset()
            creationRequest.addResource(with: .photo, data: photoData, options: nil)
        } completionHandler: { success, error in
            if let error {
                print("Error saving photo: \(error.localizedDescription)")
                return
```

Saving Photos (To Documents)

Track Results Through a Photo Capture Delegate

The method `capturePhoto` only begins the process of taking a photo. The rest of the process happens in delegate methods that the app implements.



We can pass in an @escaping function inside didFinishProcessingPhoto in order to capture the photo output and save it in the image folder of our choosing

Capturing photo inside didFinishProcessingPhoto

```swift
class PhotoCaptureManager {
    let photoOutput: AVCapturePhotoOutput
    private var photoCaptureDelegateContainer: Set<PhotoCaptureProcessor>

    init() {
        photoOutput = AVCapturePhotoOutput.createConfiguredPhotoOutput()
        photoCaptureDelegateContainer = Set<PhotoCaptureProcessor>()
    }

    // MARK: Capture & Save Photo
    func saveFrame(to path: URL?) {
        // Setup delegate & AVcapturePhotoSettings
        let photoCaptureSettings = AVCapturePhotoSettings.createConfiguredAVCapturePhotoSettings(format: self.photoOutput.appropriatePhotoCodecType())
        let photoCaptureDelegate = PhotoCaptureProcessor(with: { [weak self] photo in self?.save(photo, at: path) })
        photoCaptureDelegate.completionHandler = { [weak self] in self?.updateDelegateContainer(for: photoCaptureDelegate) }

        // Add photoCaptureDelegate to the container to hold reference so that it doesn't get garbage collected
        updateDelegateContainer(for: photoCaptureDelegate)
        Task { self.photoOutput.capturePhoto(with: photoCaptureSettings, delegate: photoCaptureDelegate) }
    }

    func save(_ photo: AVCapturePhotoPhoto, at path: URL?) {...}

    func updateDelegateContainer(for delegate: PhotoCaptureProcessor) {...}
}
```

AVCapturePhoto

A container for image data from a photo capture output.

iOS 11.0+ | iPadOS 11.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class AVCapturePhoto : NSObject
```

Overview

When you capture photos with the `AVCapturePhotoOutput` class, your delegate object receives each resulting image and related data in the form of an `AVCapturePhoto` object. This object is an immutable wrapper from which you can retrieve various results of the photo capture.

In addition to the photo image pixel buffer, an AVCapturePhoto object can also contain a preview-sized pixel buffer, capture metadata, and, on supported devices, depth data and camera calibration data. From an `AVCapturePhoto` object, you can generate data appropriate for writing to a file, such as HEVC encoded image data containerized in the HEIC file format and including a preview image, depth data and other attachments.

An `AVCapturePhoto` instance wraps a single image result. For example, if you request a bracketed capture of three images, your callback is called three times, each time delivering a single `AVCapturePhoto` object.

```swift
// MARK: Receiving Capture Results
func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVCapturePhoto, error: Error?) {
    // Where captured photo should be caught and saved
    print("Finished Processing Photo!")
    Task {
        photoProcessingHandler(photo)
        await MainActor.run { completionHandler() }
    }
}
```

AVCapturePhoto Instance Method

AVCapturePhoto

A container for image data from a photo capture output.

iOS 11.0+ | iPadOS 11.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
class AVCapturePhoto : NSObject
```

Overview

When you capture photos with the `AVCapturePhotoOutput` class, your delegate object receives each resulting image and related data in the form of an `AVCapturePhoto` object. This object is an immutable wrapper from which you can retrieve various results of the photo capture.

In addition to the photo image pixel buffer, an AVCapturePhoto object can also contain a preview-sized pixel buffer, capture metadata, and, on supported devices, depth data and camera calibration data. From an `AVCapturePhoto` object, you can generate data appropriate for writing to a file, such as HEVC encoded image data containerized in the HEIC file format and including a preview image, depth data and other attachments.

An `AVCapturePhoto` instance wraps a single image result. For example, if you request a bracketed capture of three images, your callback is called three times, each time delivering a single `AVCapturePhoto` object.

Instance Method

fileDataRepresentation()

Generates and returns a flat data representation of the photo and its attachments.

iOS 11.0+ | iPadOS 11.0+ | macOS 10.15+ | Mac Catalyst 14.0+ | tvOS 17.0+

```
func fileDataRepresentation() -> Data?
```

Return Value

Data appropriate for writing to a file of the type specified when requesting photo capture, or `nil` if the photo and attachment data cannot be flattened.

Discussion

When you request a photo capture with the `AVCapturePhotoOutput capturePhoto(with:delegate:)` method, the `AVCapturePhotoSettings` object you provide specifies image data formats (such as JPEG and HEVC) and container file formats (such as JFIF and HEIF) for the resulting image file. Calling this method formats and packages the image pixel buffer, along with metadata and other attachments created during capture (such as preview photos and depth maps), into data appropriate for writing to a file of that type.

Calling fileDataRepresentation() method on AVCapturePhoto type turns the AVCapturePhoto type to Data type, with which we can call the atomic write operation of the said data, as will be shown in the next slide.

Saving the Data transformed from AVCapturePhoto using data.write atomically

```swift
class DirectoryManager {
    let dirPath: URL?
    let filePrefix: String
    let fileSuffix: String
    var numPhotos: UInt32
    var nextImagePath: URL? { dirPath?.appendingPathComponent(String(format: "%@%04d", filePrefix, numPhotos).appending(fileSuffix)) }

    init(filePrefixInDirectory: String,
         fileSuffixInDirectory: String) {
        filePrefix = filePrefixInDirectory
        numPhotos = 0
        dirPath = DirectoryManager.createNewDirectory()
        fileSuffix = fileSuffixInDirectory
    }

    /// This should be done atomically
    static func createFile(at imageURL: URL?, contents data: Data?) {
        if let imageURL = imageURL, let data = data {
            do {
                try data.write(to: URL(fileURLWithPath: imageURL.path), options: .atomic)
            }
            catch { print("Couldn't write image data to file") }
        }
        else { print("Image path (or data) was nil") }
    }

    static func createNewDirectory() -> URL? {
        if let appCapturesFolder = DirectoryManager.appCapturesFolder() {
            let directoryPath = appCapturesFolder.appendingPathComponent(DirectoryManager.returnTimeStamp(), isDirectory: true)
            do {
                try FileManager.default.createDirectory(atPath: directoryPath.path, withIntermediateDirectories: true)
                return directoryPath
            } catch {
                print("Error \(error.localizedDescription) occurred")
                fatalError("Image directory could not be created")
            }
        } else {
            return nil
        }
    }
```

write(to:options:)

Writes the contents of the data buffer to a location.

iOS 8.0+ | iPadOS 8.0+ | macOS 10.10+ | Mac Catalyst 13.0+ | tvOS 9.0+ | watchOS 2.0+ | visionOS 1.0+ | Xcode 8.0+

```swift
func write(
    to url: URL,
    options: Data.WritingOptions = []
) throws
```

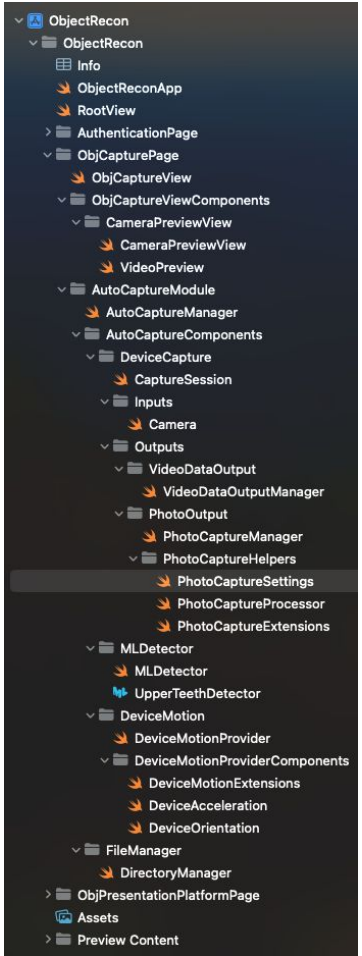
Parameters

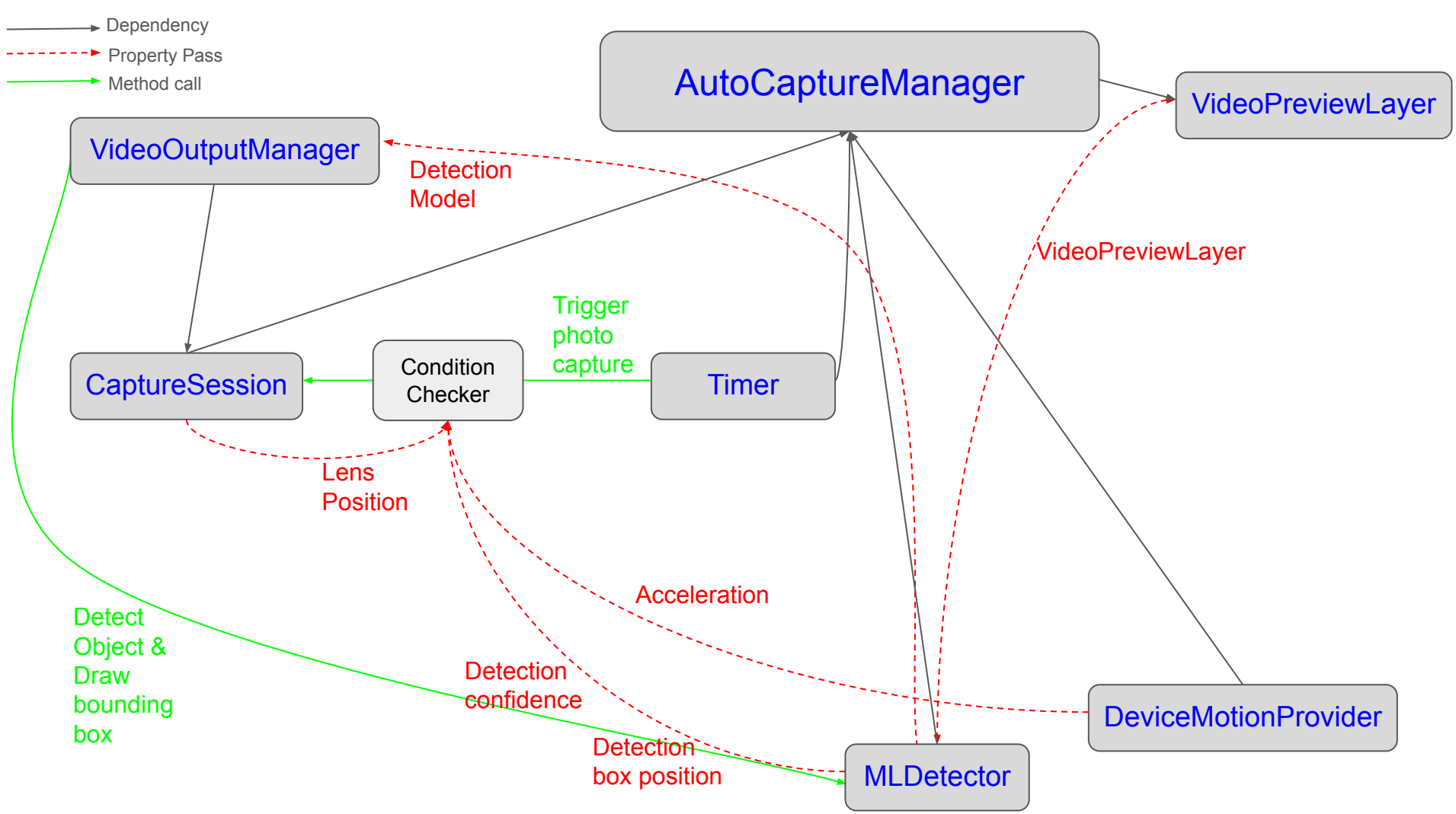
url
The location to write the data into.

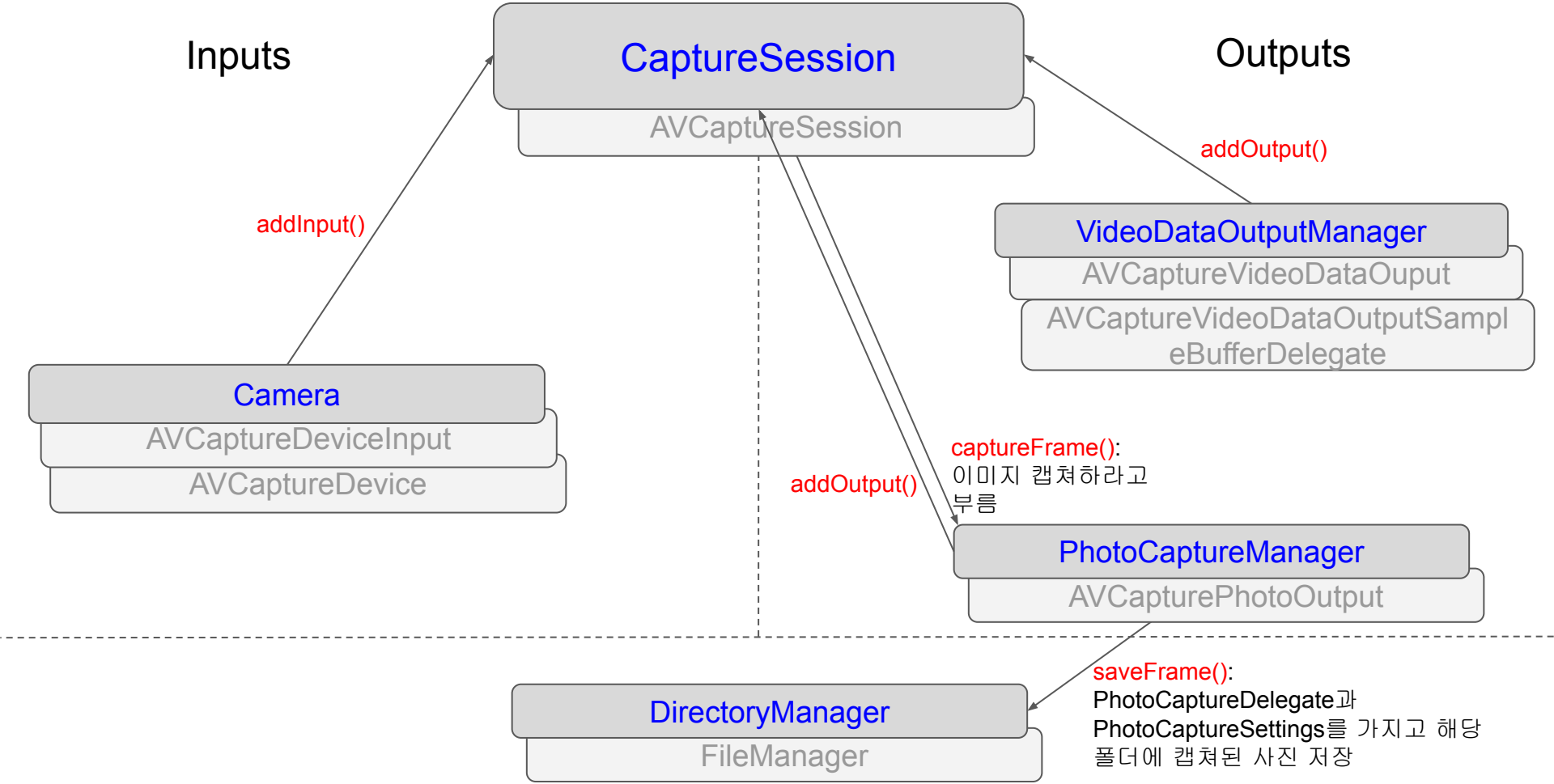
options
Options for writing the data. Default value is [].

Our Implementation

Folder Structure




Dependency
Property Pass
Method call
AutoCaptureManager
VideoPreviewLayer
VideoOutputManager
Detection Model
VideoPreviewLayer
Trigger photo capture
CaptureSession
Condition Checker
Timer
Lens Position
Acceleration
Detect Object & Draw bounding box
Detection confidence
DeviceMotionProvider
Detection box position
MLDetector


Inputs
CaptureSession
AVCaptureSession
Outputs
addInput()
addOutput()
VideoDataOutputManager
AVCaptureVideoDataOuput
AVCaptureVideoDataOutputSampleBufferDelegate
Camera
AVCaptureDeviceInput
AVCaptureDevice
addOutput()
captureFrame():
이미지 캡쳐하라고
부름
PhotoCaptureManager
AVCapturePhotoOutput
saveFrame():
PhotoCaptureDelegate과
PhotoCaptureSettings를 가지고 해당
폴더에 캡쳐된 사진 저장
DirectoryManager
FileManager