Working with HEIF and HEVC

Session 511

Erik Turnquist, CoreMedia Engineer
Brad Ford, Camera Software
What is HEVC?
Why?
Up to 40%

Better compression than H.264
Up to 2x

Better compression
Today
Access
Playback
Capture
Export
Access
Playback
Capture
Export
HEVC Assets From Photos

PhotoKit will deliver HEVC assets for playback

// PHImageManager
manager.requestPlayerItem(forVideo: asset, options: nil) { (playerItem, dictionary) in
    // use AVPlayerItem
}

manager.requestLivePhoto(for: asset, targetSize: size, contentMode: .default, options: nil) { (livePhoto, dictionary) in
    // use PHLivePhoto
}
HEVC Assets From Photos

PhotoKit will deliver HEVC assets

```swift
// PHImageManager
manager.requestExportSession(forVideo: asset, options: nil, exportPreset: preset) { (session, dictionary) in
    // use AVAssetExportSession
}
manager.requestAVAsset(forVideo: asset, options: nil) { (asset, audioMix, dictionary) in
    // use AVAsset
}
```
HEVC Assets From Photos

Access HEVC movie file data

```swift
// PHAssetResourceManager
resourceManager.requestData(for: assetResource, options: nil, dataReceivedHandler: { (data) in
    // use Data
}, { (error) in
    // handle Error
})
```
Access
Playback
Capture
Export
Native Playback Support

Supported in modern media frameworks

Streaming, play-while-download, and local files are supported

MPEG-4, QuickTime file format container support

No API opt-in required
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>macOS</td>
<td>[✓]</td>
</tr>
<tr>
<td>iOS</td>
<td>[✓]</td>
</tr>
</tbody>
</table>
## HEVC Decode Support

**Minimum configurations**

<table>
<thead>
<tr>
<th></th>
<th>iOS</th>
<th>macOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit Hardware Decode</td>
<td>A9 chip</td>
<td>6th Generation Intel Core processor</td>
</tr>
<tr>
<td>10-bit Hardware Decode</td>
<td></td>
<td>7th Generation Intel Core processor</td>
</tr>
<tr>
<td>8-bit Software Decode</td>
<td>All iOS Devices</td>
<td>All Macs</td>
</tr>
<tr>
<td>10-bit Software Decode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
let player = AVPlayer(url: URL(fileURLWithPath: "MyAwesomeMovie.mov"))
player.play()}
Decode Capability

Useful for non-realtime operations

Can be limited by hardware support

assetTrack.isDecodable
Playback Capability

Not all content can be played back in realtime

Differing capabilities on device

assetTrack.isPlayable
Hardware Decode Availability

Longest battery life

Best decode performance

```
let hardwareDecodeSupported = VTIsHardwareDecodeSupported(kCMVideoCodecType_HEVC)
```
<table>
<thead>
<tr>
<th>Feature</th>
<th>H.264</th>
<th>HEVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most Compatible</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Smaller file size</td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>
Access
Playback
Capture
Export
Capture HEVC movies with AVFoundation

MPEG-4, QuickTime file format container support
# HEVC Capture Support

## Minimum configurations

| iOS   |  
|-------|---
| 8-Bit Hardware Encode | A10 Fusion chip |
Capturing Movies with HEVC
Capturing Movies with HEVC

AVCaptureSession
Capturing Movies with HEVC

AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession
Capturing Movies with HEVC

AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

AVCaptureMovieFileOutput
Capturing Movies with HEVC

AVCaptureDevice

AVCaptureDeviceInput

AVCaptureConnection

AVCaptureSession

AVCaptureMovieFileOutput
Capturing Movies with HEVC

- AVCaptureDevice
- AVCaptureDeviceInput
- AVCaptureConnection
- AVCaptureSession
- AVCaptureMovieFileOutput

MOV
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```swift
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtinWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```swift
let session = AVCaptureSession()
session.sessionPreset = .hd4K3840x2160

let camera = AVCaptureDevice.default(.builtInWideAngleCamera, for: nil, position: .back)
let input = try! AVCaptureDeviceInput(device: camera!)
session.addInput(input)

let movieFileOutput = AVCaptureMovieFileOutput()
session.addOutput(movieFileOutput)

session.startRunning()
movieFileOutput.startRecording(to: url, recordingDelegate: self)
```
let connection = movieFileOutput.connection(with: .video)
if movieFileOutput.availableVideoCodecTypes.contains(.hevc) {
    outputSettings = [AVVideoCodecKey: AVVideoCodecType.hevc]
} else {
    outputSettings = [AVVideoCodecKey: AVVideoCodecType.h264]
}

movieFileOutput.setOutputSettings(outputSettings, for: connection!)
let connection = movieFileOutput.connection(with: .video)

if movieFileOutput.availableVideoCodecTypes.contains(.hevc) {
    outputSettings = [AVVideoCodecKey: AVVideoCodecType.hevc]
} else {
    outputSettings = [AVVideoCodecKey: AVVideoCodecType.h264]
}

movieFileOutput.setOutputSettings(outputSettings, for: connection!)
let connection = movieFileOutput.connection(with: .video)
if movieFileOutput.availableVideoCodecTypes.contains(.hevc) {
    outputSettings = [AVVideoCodecKey: AVVideoCodecType.hevc]
} else {
    outputSettings = [AVVideoCodecKey: AVVideoCodecType.h264]
}
movieFileOutput.setOutputSettings(outputSettings, for: connection!)
Capturing Live Photo Movies with HEVC
Capturing Live Photo Movies with HEVC

AVCaptureDevice

AVCaptureDeviceInput

AVCaptureSession

AVCapturePhotoOutput

MOV
Live Photo Enhancements

Video stabilization

Music playback

30 FPS
let photoSettings = AVCapturePhotoSettings()
photoSettings.livePhotoMovieFileURL = URL(fileURLWithPath: myFilePath)
if photoOutput.availableLivePhotoVideoCodecTypes.contains(.hevc) {
    photoSettings.livePhotoVideoCodecType = .hevc
}
photoOutput.capturePhoto(with: photoSettings, delegate: self)
let photoSettings = AVCapturePhotoSettings()

photoSettings.livePhotoMovieFileURL = URL(fileURLWithPath: myFilePath)

if photoOutput.availableLivePhotoVideoCodecTypes.contains(.hevc) {
    photoSettings.livePhotoVideoCodecType = .hevc
}

photoOutput.capturePhoto(with: photoSettings, delegate: self)
let photoSettings = AVCapturePhotoSettings()
photoSettings.livePhotoMovieFileURL = URL(fileURLWithPath: myFilePath)
if photoOutput.availableLivePhotoVideoCodecTypes.contains(.hevc) {
    photoSettings.livePhotoVideoCodecType = .hevc
}
photoOutput.capturePhoto(with: photoSettings, delegate: self)
Capturing HEVC Movies with AVAssetWriter
Capturing HEVC Movies with AVAssetWriter

Diagram:
- AVCaptureDevice
  - AVCaptureDeviceInput
    - AVCaptureSession
      - AVCaptureVideoDataOutput
        - sbuf
          - AVAssetWriter
            - MOV
Capturing HEVC Movies with AVAssetWriter

Configure AVAssetWriterInput with output settings

Video data output can recommend settings

// iOS 7
vdo.recommendedVideoSettingsForAssetWriter(writingTo: .mov)

// iOS 11
vdo.recommendedVideoSettings(forVideoCodecType: .hevc, assetWriterOutputFileType: .mov)
Capturing HEVC Movies with AVAssetWriter

Configure AVAssetWriterInput with output settings

Video data output can recommend settings

// iOS 7
vdo.recommendedVideoSettingsForAssetWriter(writingTo: .mov)

// iOS 11
vdo.recommendedVideoSettings(forVideoCodecType: .hevc, assetWriterOutputFileType: .mov)
Export

Transcode to HEVC with AVFoundation and VideoToolbox

MPEG-4, QuickTime file format container support

API opt-in required
<table>
<thead>
<tr>
<th>Platform</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>macOS</td>
<td>✅</td>
</tr>
<tr>
<td>iOS</td>
<td>✅</td>
</tr>
</tbody>
</table>
## HEVC Encode Support

**Minimum configurations**

<table>
<thead>
<tr>
<th></th>
<th>iOS</th>
<th>macOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit Hardware Encode</td>
<td>A10 Fusion chip</td>
<td>6th Generation Intel Core processor</td>
</tr>
<tr>
<td>10-bit Software Encode</td>
<td></td>
<td>All Macs</td>
</tr>
</tbody>
</table>
Transcode with AVAssetExportSession
Transcode with AVAssetExportSession
Export Session

No change in behavior for existing presets

Convert from H.264 to HEVC with new presets

Produce smaller AVAssets with same quality

AVAssetExportPresetHEVC1920x1080
AVAssetExportPresetHEVC3840x2160
AVAssetExportPresetHEVCHighestQuality
Compression with AVAssetWriter
Compression with AVAssetWriter
Specify HEVC with output settings for AVAssetWriterInput

```swift
settings = [AVVideoCodecKey: AVVideoCodecType.hevc]
```

Convenient output settings with AVOutputSettingsAssistant

- AVOutputSettingsPreset.hevc1920x1080
- AVOutputSettingsPreset.hevc3840x2160
Valid Output Settings

Query encoder for supported properties in output settings

```swift
let error = VTCopySupportedPropertyDictionaryForEncoder(
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification,
    &encoderID, &properties)

if error == kVTCouldNotFindVideoEncoderErr {
    // no HEVC encoder
}
```

Encoder ID is a unique identifier for an encoder

Properties and encoder ID can be used in output settings
Valid Output Settings

Query encoder for supported properties in output settings

```swift
let error = VTCopySupportedPropertyDictionaryForEncoder(
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification,
    &encoderID, &properties)

if error == kVTCouldNotFindVideoEncoderErr {
    // no HEVC encoder
}
```

Encoder ID is a unique identifier for an encoder

Properties and encoder ID can be used in output settings
Compress Samples with VTCompressionSession
Compress Samples with VTCompressionSession

VTCompressionSession

sbuf

hvc1
Create session with HEVC encoder

```swift
let error = VTCompressionSessionCreate(
    kCFAAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_H264,
    encoderSpecification,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler &session);

if error == kVTCouldNotFindVideoEncoderErr {
    // no H.264 encoder
}
```
Compression Session

Create session with HEVC encoder

```swift
let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification,
    nil, nil, nil, nil) // using VTCompressionSessionEncodeFrameWithOutputHandler &session);

if error == kVTCouldNotFindVideoEncoderErr {
    // no HEVC encoder
}
```
// Use hardware when available on macOS

let encoderSpecification: [CFString: Any] = [
  kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
  kCFAllocatorDefault,
  3840, 2160,
  kCMVideoCodecType_HEVC,
  encoderSpecification as CFDictionary,
  nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
  &session)

// Using hardware, or software
let encoderSpecification: [CFString: Any] = [
  kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
  kCFAAllocatorDefault,
  3840, 2160,
  kCMVideoCodecType_HEVC,
  encoderSpecification as CFDictionary,
  nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
  &session)

// Using hardware, or software
// Use hardware when available on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

// Using hardware, or software
// Realtime encode, fail if no hardware exists on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_RequireHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler
    &session)

if error == kVTCouldNotFindVideoEncoderErr {
    // no hardware HEVC encoder
}
// Realtime encode, fail if no hardware exists on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_RequireHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAlocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler &session
)

if error == kVTCouldNotFindVideoEncoderErr {
    // no hardware HEVC encoder
}
// Realtime encode, fail if no hardware exists on macOS

let encoderSpecification: [CFString: Any] = [
    kVTVideoEncoderSpecification_RequireHardwareAcceleratedVideoEncoder: true
]

let error = VTCompressionSessionCreate(
    kCFAllocatorDefault,
    3840, 2160,
    kCMVideoCodecType_HEVC,
    encoderSpecification as CFDictionary,
    nil, nil, nil, nil, // using VTCompressionSessionEncodeFrameWithOutputHandler &session
)

if error == kVTCouldNotFindVideoEncoderErr {
    // no hardware HEVC encoder
}
Bit Depth
Bit depth

8-bit*

10-bit

*Effect amplified for illustration
HEVC 10-bit Encode Settings

Set profile via `kVTCompressionPropertyKey_ProfileLevel`

```c
// Check VTSessionCopySupportedPropertyDictionary() for support
kVTProfileLevel_HEVC_Main10_AutoLevel
```

CoreVideo pixel buffer format

```c
kCVPixelFormatType_420YpCbCr10BiPlanarVideoRange // 10-bit 4:2:0
```
HEVC 10-bit Encode Settings

Set profile via kVTCompressionPropertyKey_ProfileLevel

// Check VTSessionCopySupportedPropertyDictionary() for support
kVTProfileLevel_HEVC_Main10_AutoLevel

CoreVideo pixel buffer format

kCVPixelFormatType_420YpCbCr10BiPlanarVideoRange // 10-bit 4:2:0
Hierarchical Frame Encoding
Video Encoding 101
Video Encoding 101

I Frame
Video Encoding 101

I Frame

P Frame
Video Encoding 101

I Frame

P Frame

B Frame
Video Encoding 101

I Frame

P Frame

B Frame

Can drop a frame when another doesn’t depend on it
Video Encoding 101

- **I Frame**
- **P Frame**
- **B Frame**

Can drop a frame when another doesn’t depend on it
Video Encoding 101

- **I Frame**
- **P Frame**
- **B Frame**

Can drop a frame when another doesn’t depend on it.
Compatible High Frame Rate Content

Droppable

Non-droppable
Compatible High Frame Rate Content

Droppable

Non-droppable

240 fps
Compatible High Frame Rate Content

Droppable

Non-droppable

120 fps
Compatible High Frame Rate Content

Droppable

Non-droppable

60 fps
Compatible High Frame Rate Content

Droppable

Non-droppable

30 fps
<table>
<thead>
<tr>
<th>Temporal Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
</tr>
<tr>
<td>Level 2</td>
</tr>
<tr>
<td>Level 1</td>
</tr>
<tr>
<td>Level 0</td>
</tr>
</tbody>
</table>
Temporal Levels

Level 0: 240 fps

Level 1

Level 2

Level 3
Temporal Levels

Level 0

Level 1

Level 2

Level 3

240 fps
Temporal Levels

Level 0

Level 1

Level 2

Level 3

30 fps
HEVC Hierarchical Encoding

Improved temporal scalability

Improved motion compensation

File annotations (MPEG-4 Part 15 - 8.4)
Create compatible high frame rate content

Set base layer and capture frame rate

// Check VTSessionCopySupportedPropertyDictionary() for support
kVTCompressionPropertyKey_BaseLayerFrameRate // temporal level 0 frame rate
kVTCompressionPropertyKey_ExpectedFrameRate // frame rate of content

Base layer must be decoded

Decode or drop other levels
HEIF — it’s what’s for dinner
What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files
What is HEIF?

Low-level HEIF file access
High-level HEIF file access
Capturing HEIF files
What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files
What is HEIF?

Low-level HEIF file access

High-level HEIF file access

Capturing HEIF files
What is HEIF?
Low-level HEIF file access
High-level HEIF file access
Capturing HEIF files
What is HEIF?
High Efficiency Image File (Format)
HEIF Pronunciation

- Hife 23%
- Heef 46%
- Heff 14%
- Eff 9%
- Хейф 7%
Why HEIF?
Why HEIF?
Why HEIF?

2x SMALLER
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?
Why HEIF?

EXIF

XMP
Why HEIF?
Demo
Extreme zooming with HEIF
# What the .HEIC?

<table>
<thead>
<tr>
<th>Payload</th>
<th>Extension</th>
<th>UTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEVC Image</td>
<td>.HEIC</td>
<td>&quot;public.heic&quot;</td>
</tr>
<tr>
<td>H.264 Image</td>
<td>.AVCI</td>
<td>&quot;public.avci&quot;</td>
</tr>
<tr>
<td>Anything Else</td>
<td>.HEIF</td>
<td>&quot;public.heif&quot;</td>
</tr>
</tbody>
</table>
# What the .HEIC?

<table>
<thead>
<tr>
<th>Payload</th>
<th>Extension</th>
<th>UTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEVC Image</td>
<td>.HEIC</td>
<td>&quot;public.heic&quot;</td>
</tr>
<tr>
<td>H.264 Image</td>
<td>.AVCI</td>
<td>&quot;public.avci&quot;</td>
</tr>
<tr>
<td>Anything Else</td>
<td>.HEIF</td>
<td>&quot;public.heif&quot;</td>
</tr>
</tbody>
</table>
## What the .HEIC?

<table>
<thead>
<tr>
<th>Payload</th>
<th>Extension</th>
<th>UTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEVC Image</td>
<td>.HEIC</td>
<td>&quot;public.heic&quot;</td>
</tr>
<tr>
<td>H.264 Image</td>
<td>.AVCI</td>
<td>&quot;public.avci&quot;</td>
</tr>
<tr>
<td>Anything Else</td>
<td>.HEIF</td>
<td>&quot;public.heif&quot;</td>
</tr>
</tbody>
</table>
### What the .HEIC?

<table>
<thead>
<tr>
<th>Payload</th>
<th>Extension</th>
<th>UTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEVC Image</td>
<td>.HEIC</td>
<td>&quot;public.heic&quot;</td>
</tr>
<tr>
<td>H.264 Image</td>
<td>.AVCI</td>
<td>&quot;public.avci&quot;</td>
</tr>
<tr>
<td>Anything Else</td>
<td>.HEIF</td>
<td>&quot;public.heif&quot;</td>
</tr>
</tbody>
</table>
## Supported HEIF Flavors (Writing)

<table>
<thead>
<tr>
<th>Payload</th>
<th>Extension</th>
<th>UTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEVC Image</td>
<td>.HEIC</td>
<td>&quot;public.heic&quot;</td>
</tr>
</tbody>
</table>
Low-Level Access to HEIF
UIKit
CoreImage
CoreGraphics
ImageIO
// Read a jpeg image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
   kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
// Read a jpeg image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
// Read a jpeg image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.jpg")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
// Read a heic image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.heic")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true, kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
let inputURL = URL(fileURLWithPath: "/tmp/image.heic")
let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]
let image = CGImageSourceCreateImageAtIndex(source, 0, nil)
// Read a heic image from file

let inputURL = URL(fileURLWithPath: "/tmp/image.heic")

let source = CGImageSourceCreateWithURL(inputURL as CFURL, nil)

let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

let image = CGImageSourceCreateImageAtIndex(source, 0, nil)

let options = [kCGImageSourceCreateThumbnailFromImageIfAbsent as String: true,
               kCGImageSourceThumbnailMaxPixelSize as String: 320] as [String: Any]

let thumb = CGImageSourceCreateThumbnailAtIndex(source, 0, options as CFDictionary)
Tiling Support in CGImageSource

```swift
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

"{TIFF}" = {
    DateTime = "2017:04:01 22:50:24";
    Make = Apple;
    Model = "iPhone 7 Plus";
    Orientation = 1;
    ResolutionUnit = 2;
    Software = "11.0";
    TileLength = 512;
    TileWidth = 512;
    XResolution = 72;
    YResolution = 72;
};
```
Tiling Support in CGImageSource

```swift
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

"{TIFF}" = {
    DateTime = "2017:04:01 22:50:24";
    Make = Apple;
    Model = "iPhone 7 Plus";
    Orientation = 1;
    ResolutionUnit = 2;
    Software = "11.0";
    TileLength = 512;
    TileWidth = 512;
    XResolution = 72;
    YResolution = 72;
};
```
Tiling Support in CGImageSource

```swift
let imageProperties = CGImageSourceCopyPropertiesAtIndex(source, 0, nil) as? [String: Any]

"{TIFF}" = {
    DateTime = "2017:04:01 22:50:24";
    Make = Apple;
    Model = "iPhone 7 Plus";
    Orientation = 1;
    ResolutionUnit = 2;
    Software = "11.0";
    TileLength = 512;
    TileWidth = 512;
    XResolution = 72;
    YResolution = 72;
};
```
Tiling Support in CGImage

```swift
let subImage = bigImage.cropping(to: rect)
```
let subImage = bigImage.cropping(to: rect)
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
    AVFileType.jpg as CFString, 1, nil) else {
    fatalError("unable to create CGImageDestination")
}

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")

guard let destination = CGImageDestinationCreateWithURL(url as CFURL, AVFileType.jpg as CFString, 1, nil) else {
    fatalError("unable to create CGImageDestination")
}

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
        AVFileType.jpg as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
    }

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
// Writing a CGImage to a JPEG file

let url = URL(fileURLWithPath: "/tmp/output.jpg")

guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
                        AVFileType.jpg as CFString, 1, nil)
else {
    fatalError("unable to create CGImageDestination")
}

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
// Writing a CGImage to a HEIC file

let url = URL(fileURLWithPath: "/tmp/output.heic")

guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
            AVFileType.heic as CFString, 1, nil)

else {
    fatalError("unable to create CGImageDestination")
}

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
Writing a CGImage to a HEIF file

```swift
let url = URL(fileURLWithPath: "/tmp/output.heic"

guard let destination = CGImageDestinationCreateWithURL(url as CFURL, AVFileType.heic as CFString, 1, nil) {
    // Add image
    CGImageDestinationAddImage(destination, image, nil)
} else {
    throw WriteError.error(message: "unable to create CGImageDestination")
}

CGImageDestinationFinalize(destination)
```
/ Writing a CGImage to a HEIF file

let url = URL(fileURLWithPath: "/tmp/output.heic")
guard let destination = CGImageDestinationCreateWithURL(url as CFURL,
        AVFileType.heic as CFString, 1, nil)
else {
    fatalError("unable to create CGImageDestination")
}

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
// Writing a CGImage to a HEIF file

let url = URL(fileURLWithPath: "/tmp/output.heic")

guard let destination = CGImageDestinationCreateWithURL(url as CFURL, AVFileType.heic as CFString, 1, nil)
    else {
        fatalError("unable to create CGImageDestination")
}

CGImageDestinationAddImage(imageDestination, image, nil)

CGImageDestinationFinalize(imageDestination)
ImageIO Depth Support

HEIC
• Auxiliary image (monochrome HEVC) with XMP metadata

JPEG
• MPO image (jpeg encoded) with XMP metadata
High-Level Access to HEIF
Compression Poetry

Brad Ford
JPEG is yay big
but
HEIF is brief
Compression Haiku

Brad Ford
HEVC has
HEVC has twice as many syllables
HEVC has twice as many syllables as JPEG — progress.
HEIF and PhotoKit

PhotosUI
Photos
UIKit
CoreImage
AVFoundation
CoreGraphics
CoreMedia
HEIF and PhotoKit

Applying adjustments
• Photos
• Videos
• Live Photos

Common workflows
• Display
• Backup
• Share
Apply Adjustments

PHPhotoLibrary performChanges:
Apply Adjustments

PHPhotoLibrary performChanges:

PHAsset
Apply Adjustments

PHPhotoLibrary performChanges:

- PHAsset

PHContentEditingInput
Apply Adjustments

```
PHPhotoLibrary performChanges:
  PHAsset
  PHContentEditingInput
  PHContentEditingOutput
```
Apply Adjustments

PHPhotoLibrary performChanges:

PHAsset

PHContentEditingInput

PHContentEditingOutput

renderedContentURL
Apply Adjustments

PHPhotoLibrary performChanges:

PHAsset

PHContentEditingInput

PHContentEditingOutput

renderedContentURL
Applying Adjustments

PHContentEditingOutput
• Image rendered as JPEG
• Image Exif orientation of 1 (no rotation)
/ Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completion: () -> ()) {

    guard let inputImage = CIImage(contentsOf: input.fullSizeImageURL!) else { fatalError("can't load input image") }

    let outputImage = inputImage.applyingOrientation(input.fullSizeImageOrientation)
    .applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage,
      to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)") }

    completion()
}
// Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completion: () -> ()) {
    guard let inputImage = CIImage(contentsOf: input.fullSizeImageURL!) else { fatalError("can't load input image") }
    let outputImage = inputImage.applyingOrientation(input.fullSizeImageOrientation).applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage, to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)") }
    completion()
}
// Editing a HEIF photo -- save as JPEG

```swift
func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completion: () -> ()) {
    guard let inputImage = CIImage(contentsOf: input.fullSizeImageURL!) else { fatalError("can't load input image") }
    let outputImage = inputImage.applyingOrientation(input.fullSizeImageOrientation).applyingFilter(filterName, withInputParameters: nil)

    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage, to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:]) } catch let error { fatalError("can't apply filter to image: \(error)") }
    completion()
}
```
// Editing a HEIF photo -- save as JPEG

func applyPhotoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completion: () -> ()) {
    guard let inputImage = CIImage(contentsOf: input.fullSizeImageURL!) else { fatalError("can't load input image") }
    let outputImage = inputImage.applyingOrientation(input.fullSizeImageOrientation).applyingFilter(filterName, withInputParameters: nil)
    // Write the edited image as a JPEG.
    do { try self.ciContext.writeJPEGRepresentation(of: outputImage, to: output.renderedContentURL, colorSpace: inputImage.colorSpace!, options: [:])
    } catch let error { fatalError("can't apply filter to image: \(error)"") }
    completion()
}
Applying Adjustments

PHContentEditingOutput

• Video rendered as H.264
// Editing an HEVC video -- save as H.264

func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completionHandler: @escaping () -> ()) {

    guard let avAsset = input.audiovisualAsset
    else { fatalError("can't get AV asset") }

    let composition = AVVideoComposition(asset: avAsset, applyingCIFiltersWithHandler: { request in
        let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
        request.finish(with: img, context: nil)
    })

    // Export the video composition to the output URL.
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality) {
        export.outputFileType = AVFileType.mov
        export.outputURL = output.renderedContentURL
        export.videoComposition = composition
        export.exportAsynchronously(completionHandler: completionHandler)
    } else { fatalError("can't set up AV export session") }
}
func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completionHandler: @escaping () -> ()) {
    guard let avAsset = input.audiovisualAsset
        else { fatalError("can't get AV asset") }

    let composition = AVVideoComposition(asset: avAsset,
    applyingCIFiltersWithHandler: { request in
        let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
        request.finish(with: img, context: nil)
    })

    // Export the video composition to the output URL.
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality)
        else { fatalError("can't set up AV export session") }
    export.outputFileType = AVFileType.mov
    export.outputURL = output.renderedContentURL
    export.videoComposition = composition
    export.exportAsynchronously(completionHandler: completionHandler)
}
func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completionHandler: @escaping () -> ()) {

    guard let avAsset = input.audiovisualAsset else { fatalError("can't get AV asset") }

    let composition = AVVideoComposition(asset: avAsset, applyingCIFiltersWithHandler: { request in
        let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
        request.finish(with: img, context: nil)
    })

    // Export the video composition to the output URL.
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality) else { fatalError("can't set up AV export session") }
    export.outputFileType = AVFileType.mov
    export.outputURL = output.renderedContentURL
    export.videoComposition = composition
    export.exportAsynchronously(completionHandler: completionHandler)
}
func applyVideoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completionHandler: @escaping () -> ()) {

    guard let avAsset = input.audiovisualAsset
    else { fatalError("can't get AV asset") }

    let composition = AVVideoComposition(asset: avAsset,
        applyingCIFiltersWithHandler: { request in
            let img = request.sourceImage.applyingFilter(filterName, withInputParameters: nil)
            request.finish(with: img, context: nil)
        })

    // Export the video composition to the output URL.
    guard let export = AVAssetExportSession(asset: avAsset, presetName: AVAssetExportPresetHighestQuality)
    else { fatalError("can't set up AV export session") }

    export.outputFileType = AVFileType.mov
    export.outputURL = output.renderedContentURL
    export.videoComposition = composition
    export.exportAsynchronously(completionHandler: completionHandler)
}
Applying Adjustments

PHLivePhotoEditingContext

• Uses CIImage frames, automatically converted
// Editing a HEIF/HEVC live photo -- format handled automatically

func applyLivePhotoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completion: @escaping () -> ()) {

    guard let livePhotoContext = PHLivePhotoEditingContext(livePhotoEditingInput: input) else { fatalError("can't get live photo") }

    livePhotoContext.frameProcessor = { frame, _ in
        return frame.image.applyingFilter(filterName, withInputParameters: nil)
    }

    livePhotoContext.saveLivePhoto(to: output) {
        success, error in
        if success { completion() }
        else { print("can't output live photo") }
    }
}
func applyLivePhotoFilter(_ filterName: String, input: PHContentEditingInput, output: PHContentEditingOutput, completion: @escaping () -> ()) {

    guard let livePhotoContext = PHLivePhotoEditingContext(livePhotoEditingInput: input)
    else { fatalError("can't get live photo") }

    livePhotoContext.frameProcessor = { frame, _ in
        return frame.image.applyingFilter(filterName, withInputParameters: nil)
    }

    livePhotoContext.saveLivePhoto(to: output) { success, error in
        if success { completion() }
        else { print("can't output live photo") }
    }
}
Common PhotoKit Workflows

Display

• PHImageManager for UIImage, AVPlayerItem, or PHLivePhoto
• No code changes needed
Common PhotoKit Workflows

Backup

• PHAssetResourceManager provides resources in native format
Common PhotoKit Workflows

Sharing

Compatibility

Features
Leaving the Garden

// for images, check the UTI and convert if needed:
PHImageManager.default().requestImageData(for: asset, options: options, resultHandler:

    { imageData, dataUTI, orientation, info in
      guard let dataUTI = dataUTI else { return }
      if !UTTypeConformsTo(dataUTI as CFString, kUTTypeJPEG {
        // convert the data to a JPEG representation...
      }
    }

// for videos use export preset to specify the format
PHImageManager.default().requestExportSession(forVideo: asset, options: options,
  exportPreset: AVAssetExportPresetHighestQuality, resultHandler: { exportSession, info in
Leaving the Garden

```swift
// for images, check the UTI and convert if needed:
PHImageManager.default().requestImageData(for: asset, options: options, resultHandler: {
    imageData, dataUTI, orientation, info in
    guard let dataUTI = dataUTI else { return }
    if !UTTypeConformsTo(dataUTI as CFString, kUTTypeJPEG {
        // convert the data to a JPEG representation...
    }

    // for videos use export preset to specify the format
    PHImageManager.default().requestExportSession(forVideo: asset, options: options, exportPreset: AVAssetExportPresetHighestQuality, resultHandler: { exportSession, info in
```
Capturing HEIF
Compression Haiku Two
HEIF, a container,
HEIF, a container, compresses four times better.
HEIF, a container, compresses four times better than HEVC
<table>
<thead>
<tr>
<th>Device</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone 7 Plus</td>
<td></td>
</tr>
<tr>
<td>iPhone 7</td>
<td></td>
</tr>
<tr>
<td>10.5-inch iPad Pro</td>
<td>NEW</td>
</tr>
<tr>
<td>12.9-inch iPad Pro</td>
<td>NEW</td>
</tr>
</tbody>
</table>
AVCapturePhotoOutput Usage
AVCapturePhotoOutput Usage

capturePhoto(with: delegate:)
f lashMode = .auto, preview = 1440x1440
AVCapturePhotoCaptureDelegate
AVCapturePhotoOutput Usage

- `AVCaptureResolvedPhotoSettings`
  - `flashEnabled = true`, `preview = 1440x1080`

- `AVCapturePhotoCaptureDelegate`
  - `willBeginCaptureFor`
  - `AVCaptureResolvedPhotoSettings`
    - `flashEnabled = true`, `preview = 1440x1080`
AVCapturePhotoOutput Usage

flashMode = .auto, preview = 1440x1440

AVCapturePhotoCaptureDelegate
AVCapturePhotoOutput Usage

flashMode = .auto, preview = 1440x1440
AVCapturePhotoOutput Usage

FlashMode = .auto, preview = 1440x1440
AVCapturePhotoOutput Usage

flashMode = .auto, preview = 1440x1440

AVCapturePhotoCaptureDelegate

Time

willBeginCaptureFor

willCapturePhotoFor
didCapturePhotoFor
AVCapturePhotoOutput Usage

flashMode = .auto, preview = 1440x1440

AVCapturePhotoCaptureDelegate
AVCapturePhotoOutput Usage

\[
\text{flashMode} = .\text{auto}, \text{preview} = 1440 \times 1440
\]

AVCapturePhotoCaptureDelegate

CMSampleBuffer

CMSampleBuffer

willBeginCaptureFor

willCapturePhotoFor

didCapturePhotoFor

didFinishProcessingPhoto
AVCapturePhotoOutput Usage

flashMode = .auto, preview = 1440x1440

AVCapturePhotoCaptureDelegate

willBeginCaptureFor
willCapturePhotoFor
didCapturePhotoFor
didFinishProcessingPhoto
AVCapturePhotoOutput Usage

flashMode = .auto, preview = 1440x1440

Time

willBeginCaptureFor
willCapturePhotoFor
didCapturePhotoFor
didFinishProcessingPhoto
didFinishCaptureFor

AVCapturePhotoCaptureDelegate
AVCapturePhotoOutput Usage
AVCapturePhotoOutput Usage

- AVCapturePhotoCaptureDelegate
- AVCapturePhotoSettings
- AVCapturePhotoOutput Usage

Time:

- willBeginCaptureFor
- willCapturePhotoFor
- didCapturePhotoFor
- didFinishProcessingPhoto
- didFinishCaptureFor
AVCapturePhotoOutput Usage

- AVCapturePhotoSettings
- AVCapturePhotoCaptureDelegate
- didFinishCaptureFor
- didFinishProcessingPhoto
- didFinishRawCaptureFor
- willCapturePhotoFor
- willBeginCaptureFor
AVCapturePhotoOutput Usage

AVCapturePhotoSettings

Time

willBeginCaptureFor
willCapturePhotoFor
didCapturePhotoFor
didFinishRawCaptureFor
didFinishProcessingPhoto
didFinishProcessingLivePhotoMovie
didFinishCaptureFor

AVCapturePhotoCaptureDelegate

AVCapturePhotoSettings

AVCapturePhotoOutput Usage
CMSampleBuffer vs. HEIF
CMSampleBuffer vs. HEIF

Sample buffer contains media data
CMSampleBuffer vs. HEIF

Sample buffer contains media data
HEIF contains a file structure
CMSampleBuffer vs. HEIF

Sample buffer contains media data

HEIF contains a file structure

HEVC video is not the same as HEIF HEVC
Faster CMSampleBuffer replacement
Faster CMSampleBuffer replacement

100% immutable
Faster CMSampleBuffer replacement

100% immutable

Backed by containerized data
open class AVCapturePhoto : NSObject {

    open var timestamp: CMTime { get }
    open var isRawPhoto: Bool { get }
    open var pixelBuffer: CVPixelBuffer? { get }

    open var previewPixelBuffer: CVPixelBuffer? { get }
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }

    open var metadata: [String : Any] { get }
    open var depthData: AVDepthData? { get }

}
open class AVCapturePhoto : NSObject {

    open var timestamp: CMTime { get }
    open var isRawPhoto: Bool { get }
    open var pixelBuffer: CVPixelBuffer? { get }

    open var previewPixelBuffer: CVPixelBuffer? { get }
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }

    open var metadata: [String : Any] { get }
    open var depthData: AVDepthData? { get }

}
open class AVCapturePhoto : NSObject {

    open var timestamp: CMTime { get }
    open var isRawPhoto: Bool { get }
    open var pixelBuffer: CVPixelBuffer? { get }

    open var previewPixelBuffer: CVPixelBuffer? { get }
    open var embeddedThumbnailPhotoFormat: [String : Any]? { get }

    open var metadata: [String : Any] { get }
    open var depthData: AVDepthData? { get }

}
open class AVCapturePhoto: NSObject {

    open var timestamp: CMTime { get }
    open var isRawPhoto: Bool { get }
    open var pixelBuffer: CVPixelBuffer? { get }

    open var previewPixelBuffer: CVPixelBuffer? { get }
    open var embeddedThumbnailPhotoFormat: [String: Any]? { get }

    open var metadata: [String: Any] { get }
    open var depthData: AVDepthData? { get }

}
open class AVCapturePhoto : NSObject {

open var resolvedSettings: AVCaptureResolvedPhotoSettings { get }
open var photoCount: Int { get }

open var bracketSettings: AVCaptureBracketedStillImageSettings? { get }
open var sequenceCount: Int { get }
open var lensStabilizationStatus: AVCaptureDevice.LensStabilizationStatus { get }

}`
open class AVCapturePhoto : NSObject {

    open var resolvedSettings: AVCaptureResolvedPhotoSettings { get }
    open var photoCount: Int { get }

    open var bracketSettings: AVCaptureBracketedStillImageSettings? { get }
    open var sequenceCount: Int { get }
    open var lensStabilizationStatus: AVCaptureDevice.LensStabilizationStatus { get }

}
open class AVCapturePhoto : NSObject {

    open var resolvedSettings: AVCaptureResolvedPhotoSettings { get }
    open var photoCount: Int { get }

    open var bracketSettings: AVCaptureBracketedStillImageSettings? { get }
    open var sequenceCount: Int { get }
    open var lensStabilizationStatus: AVCaptureDevice.LensStabilizationStatus { get }

}
open class AVCapturePhoto : NSObject {

    open func fileDataRepresentation() -> Data?

    open func cgImageRepresentation() -> Unmanaged<CGImage>?
    open func previewCGImageRepresentation() -> Unmanaged<CGImage>?

}
open class AVCapturePhoto : NSObject {

    open func fileDataRepresentation() -> Data?

    open func cgImageRepresentation() -> Unmanaged<CGImage>?
    open func previewCGImageRepresentation() -> Unmanaged<CGImage>?

}
open class AVCapturePhoto : NSObject {

    open func fileDataRepresentation() -> Data?

    open func cgImageRepresentation() -> Unmanaged<CGImage>?
    open func previewCGImageRepresentation() -> Unmanaged<CGImage>?

}

func photoOutput(_ output: AVCapturePhotoOutput,
    didFinishProcessingPhoto photo: AVCapturePhoto,
    error: Error?)
func photoOutput(_ output: AVCapturePhotoOutput,
    didFinishProcessingPhoto photoSampleBuffer: CMSampleBuffer?,
    previewPhoto: CMSampleBuffer?,
    resolvedSettings: AVCaptureResolvedPhotoSettings,
    bracketSettings: AVCaptureBracketedStillImageSettings?,
    error: Error?)

func photoOutput(_ output: AVCapturePhotoOutput,
    didFinishProcessingRawPhoto rawSampleBuffer: CMSampleBuffer?,
    previewPhoto: CMSampleBuffer?,
    resolvedSettings: AVCaptureResolvedPhotoSettings,
    bracketSettings: AVCaptureBracketedStillImageSettings?,
    error: Error?)
## iOS 10 AVCapturePhotoOutput Supported Formats

<table>
<thead>
<tr>
<th>Image Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compressed Formats</strong></td>
<td></td>
</tr>
<tr>
<td>jpeg</td>
<td></td>
</tr>
<tr>
<td><strong>Uncompressed Formats</strong></td>
<td></td>
</tr>
<tr>
<td>420f/420v</td>
<td></td>
</tr>
<tr>
<td>BGRA</td>
<td></td>
</tr>
<tr>
<td><strong>RAW Formats</strong></td>
<td></td>
</tr>
<tr>
<td>grb4/rgg4/bgg4/gbr4</td>
<td></td>
</tr>
</tbody>
</table>
# iOS 11 AVCapturePhotoOutput Supported Formats

<table>
<thead>
<tr>
<th></th>
<th>Image Format</th>
<th>File Container Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compressed Formats</strong></td>
<td>hvc1</td>
<td>HEIC</td>
</tr>
<tr>
<td></td>
<td>jpeg</td>
<td>JFIF</td>
</tr>
<tr>
<td><strong>Uncompressed Formats</strong></td>
<td>420f/420v</td>
<td>TIFF</td>
</tr>
<tr>
<td></td>
<td>BGRA</td>
<td></td>
</tr>
<tr>
<td><strong>RAW Formats</strong></td>
<td>grb4/rgg4/bgg4/gbr4</td>
<td>DNG</td>
</tr>
</tbody>
</table>
iOS 11 AVCapturePhotoOutput Supported Formats

<table>
<thead>
<tr>
<th>Compressed Formats</th>
<th>Image Format</th>
<th>File Container Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>hvc1</td>
<td>HEIC</td>
</tr>
<tr>
<td></td>
<td>jpeg</td>
<td>JFIF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Uncompressed Formats</th>
<th>Image Format</th>
<th>File Container Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>420f/420v</td>
<td>TIFF</td>
</tr>
<tr>
<td></td>
<td>BGRA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAW Formats</th>
<th>Image Format</th>
<th>File Container Format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>grb4/rgg4/bgg4/gbr4</td>
<td>DNG</td>
</tr>
</tbody>
</table>
Old Way

AVCapturePhotoOutput

Full Size JPEG
CMSampleBuffer

Preview 420f
CMSampleBuffer
Old Way

AVCapturePhotoOutput

Full Size JPEG
CMSampleBuffer

Preview 420f
CMSampleBuffer

→ Exif manipulation
Old Way

AVCapturePhotoOutput

Full Size JPEG CMSampleBuffer

Preview 420f CMSampleBuffer

Exif manipulation

JPEGDataPhotoRepresentation(for:, preview:)
Old Way

AVCapturePhotoOutput

Full Size JPEG
CMSampleBuffer

Preview 420f
CMSampleBuffer

Exif manipulation

JPEGDataPhotoRepresentation(for:, preview:)

JPEG
New Way
AVCapturePhotoSettings

.format = [AVVideoCodecType: .hevc]
.metadata = [...]
.previewPhotoFormat = [1440x1440]
.embeddedThumbnailPhotoFormat = [320x320]
New Way

AVCapturePhotoOutput

AVCapturePhotoSettings

.format = [AVVideoCodecType: .hevc]

.metadata = [...]

.previewPhotoFormat = [1440x1440]

.embeddedThumbnailPhotoFormat = [320x320]
New Way

AVCapturePhotoOutput

AVCapturePhotoSettings
- .format = [AVVideoCodecType: .hevc]
- .metadata = {...}
- .previewPhotoFormat = [1440x1440]
- .embeddedThumbnailPhotoFormat = [320x320]

AVCapturePhoto
(HEIC containerized)
New Way

AVCapturePhotoOutput

AVCapturePhotoSettings
- .format = [AVVideoCodecType: .hevc]
- .metadata = [...] 
- .previewPhotoFormat = [1440x1440]
- .embeddedThumbnailPhotoFormat = [320x320]

AVCapturePhoto (HEIC containerized)

photo.fileDataRepresentation()
New Way

AVCapturePhotoOutput

AVCapturePhotoSettings
- .format = [AVVideoCodecType: .hevc]
- .metadata = [...] (details not shown)
- .previewPhotoFormat = [1440x1440]
- .embeddedThumbnailPhotoFormat = [320x320]

AVCapturePhoto (HEIC containerized)

photo.loadDataRepresentation()
Photos During Video Capture

HEVC/H.264 hardware resource contention
- Video is prioritized
- Photos are larger

Consider using JPEG for photos
HEVC and HEIF Bursts

HEVC encode takes longer than JPEG

Recommendation for burst captures is to use JPEG
A Compression Essay on WWDC

Brad Ford
World Wide Developer Conference
World Wide Developer Conference
WWDC
Dub-Dub
World Wide Developer Conference
WWDC
Dub-Dub
Wuh-Duck
Summary

HEVC movies are up to 40% smaller than H.264
HEVC playback is supported everywhere on iOS 11 and macOS High Sierra
Opt in to create HEVC content using capture and export APIs
HEIC files are twice as small as JPEGs
HEIF decode is supported everywhere on iOS 11 and macOS High Sierra
Capture HEIC files using the new AVCapturePhoto interface
More Information

<table>
<thead>
<tr>
<th>Session Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Efficiency Image File Format</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Introducing HEIF and HEVC</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Depth and Capture</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Image Editing with Depth</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Lab</td>
<td>Location</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>HEIF and HEVC Lab</td>
<td>Technology Lab F</td>
</tr>
<tr>
<td>Photos Depth &amp; Capture Lab</td>
<td>Technology Lab F</td>
</tr>
</tbody>
</table>