



```

        refcon: nil,
        compressionSessionOut: &compressionSessionOut)

guard err == noErr, let compressionSession = compressionSessionOut else {
    throw RuntimeError("VTCompressionSession creation failed (\(err))!")
}

```

## Configure the compression session

Get the suggested encoder settings dictionary for the encode preset.

```

/// Get the suggested encoder settings dictionary for encode preset.
/// - Parameters:
///   - session: A compression session.
///   - encodePreset: The `EncodePreset` enumeration.
private func getEncoderSettingsForPreset(session: VTCompressionSession, encodePreset:
    var supportedPresetDictionaries: CFDictionary?
    var encoderSettings: [CFString: Any]?

    _ = withUnsafeMutablePointer(to: &supportedPresetDictionaries) { valueOut in
        VTSessionCopyProperty( session, key: kVTCompressionPropertyKey_SupportedPresets,
                               allocator: kCFAllocatorDefault, valueOut: valueOut )
    }

    if let presetDictionaries = supportedPresetDictionaries as? [CFString: [CFString: Any]] {
        let presetConstant = switch encodePreset {
        case .videoConferencing: kVTCompressionPreset_VideoConferencing
        }

        encoderSettings = presetDictionaries[presetConstant]
    }

    return encoderSettings
}

```

Set the encoder settings dictionary. Set `kVTCompressionPropertyKey_RealTime` to `kCFBooleanTrue` to indicate that this is a live encoding session. Specify the expected video source frame rate. Optionally specify the codec profile and level, the average target bit rate and the data rate limits or the target constant bit rate, the maximum interval between key frames, the number of look ahead frames, and the spatial adaptive QP level.

```

/// Configures a compression session for live streaming.
/// - Parameters:
///   - session: A compression session.
///   - options: The configuration options.
///   - expectedFrameRate: The expected frame rate of the video source.
private func configureVTCompressionSession(session: VTCompressionSession, options: C
    // Different encoder implementations may support different property sets, so
    // the app needs to determine the implications of a failed property setting
    // on a case-by-case basis for the encoder. If the property is essential for
    // the use case and its setting fails, the app terminates. Otherwise, the
    // encoder ignores the failed setting and uses a default value to proceed
    // with encoding.

    var err: OSStatus = noErr
    var variableBitRateMode = false

    if let presetTuple = options.presetTuple {
        // Try configuring the encoder using the preset.
        let encoderSettings: [CFString: Any]?
        encoderSettings = getEncoderSettingsForPreset(session: session, encodePreset

        if let encoderSettings {
            if encoderSettings[kVTCompressionPropertyKey_VariableBitRate] != nil {
                variableBitRateMode = true
            }

            // Set the encoder settings dictionary on the compression session.
            err = VTSessionSetProperties(session, propertyDictionary: encoderSetting
            try NSError.check(err, "VTSessionSetProperties failed")
        }
    }

    // Indicate real time compression session, which live streaming requires.
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_RealTime, val
    if err != noErr {
        print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_RealTime) fai
    }

    // Indicate the expected frame rate, if known. This is just a hint for rate
    // control purposes; the actual encoding frame rate matches the incoming
    // frame rate even if it doesn't match this setting. When
    // `kVTCompressionPropertyKey_RealTime` is `kCFBooleanTrue`, the video
    // encoder may optimize energy usage.

```

```

err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_ExpectedFrame
if err != noErr {
    print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_ExpectedFrame
}

// Specify the profile and level for the encoded bitstream.
if let profileTuple = options.profileTuple {
    var profileConstant: CFString?

    if options.codec == kCMVideoCodecType_H264 {
        if profileTuple.0 == .h264Main {
            profileConstant = kVTProfileLevel_H264_Main_AutoLevel
        } else if profileTuple.0 == .h264High {
            profileConstant = kVTProfileLevel_H264_High_AutoLevel
        }
    } else if options.codec == kCMVideoCodecType_HEVC {
        if profileTuple.0 == .hevcMain {
            profileConstant = kVTProfileLevel_HEVC_Main_AutoLevel
        } else if profileTuple.0 == .hevcMain10 {
            profileConstant = kVTProfileLevel_HEVC_Main10_AutoLevel
        }
    }
}

if let profileConstant {
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Profi
    if err != noErr {
        print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Profi
    }
}

if let destBitRate = options.destBitRate {
    if options.constantBitRateMode {
        // This is intended for legacy content distribution networks
        // that require constant bitrate, not for general streaming
        // scenarios. The encoder pads the frames if they are smaller
        // than necessary based on the constant bit rate.
        err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Const
        if err != noErr {
            print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Const
        }
    } else if variableBitRateMode {
        // Specify the long-term desired variable bit rate in bits per second.

```

```

err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_VariMa
if err != noErr {
    print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_VariMa
}

// Set VBV maximum bit rate.
err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_VBVMa
if err != noErr {
    print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_VBVMa
}
} else {
    // Specify the long-term desired average bit rate in bits per second.
    // It's a soft limit, so the encoder may overshoot or undershoot and
    // the average bit rate of the output video may be over or under the
    // target.
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Avera
    if err != noErr {
        print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Avera
    }

    // Specify a hard data-rate cap for a given time window, which
    // the encoder won't overshoot. Use
    // `kVTCompressionPropertyKey_AverageBitRate` and
    // `kVTCompressionPropertyKey_DataRateLimits` together to
    // specify an overall target bit rate and hard limits over a
    // smaller window.
    let byteLimit = (Double(destBitRate) / 8 * 1.5) as CFNumber
    let secLimit = Double(1.0) as CFNumber
    let limitsArray = [ byteLimit, secLimit ] as CFArray
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_DataR
    if err != noErr {
        print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_DataR
    }
}
}

if let maxKeyFrameInterval = options.maxKeyFrameInterval {
    // Specify the maximum interval between key frames, also known as
    // the key frame rate. Set this in conjunction with
    // `kVTCompressionPropertyKey_MaxKeyFrameIntervalDuration` to
    // enforce both limits, which requires a keyframe every X frames
    // or every Y seconds, whichever comes first.
    err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_MaxKeyFra

```

```

        if err != noErr {
            print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_MaxKeyFra
        }
    }

    if let maxKeyFrameIntervalDuration = options.maxKeyFrameIntervalDuration {
        // Specify the maximum duration from one key frame to the next in seconds.
        err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_MaxKeyFra
            value: maxKeyFrameIntervalDuration as CFNumber)

        if err != noErr {
            print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_MaxKeyFra
        }
    }

    if let lookAheadFrames = options.lookAheadFrames {
        // Specify the number of look ahead frames.
        err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_Suggested
        if err != noErr {
            print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_Suggested
        }
    }

    if let spatialAdaptiveQP = options.spatialAdaptiveQP {
        // Specify whether to apply spatial QP adaptation based on per-frame statist
        err = VTSessionSetProperty(session, key: kVTCompressionPropertyKey_SpatialAc
        if err != noErr {
            print("Warning: VTSessionSetProperty(kVTCompressionPropertyKey_SpatialAc
        }
    }
}
}

```

## Encode video frames

Call `VTCompressionSessionEncodeFrame(_:imageBuffer:presentationTimeStamp:duration:frameProperties:infoFlagsOut:outputHandler:)` and submit each uncompressed frame to the `VTCompressionSession` object for encoding. The object calls the `outputHandler` block for each encoded frame. Check whether a frame drop or error occurs after frame encoding.

Call `VTCompressionSessionCompleteFrames(_:untilPresentationTimeStamp:)` to indicate to the `VTCompressionSession` object that the app submitted all uncompressed video frames for encoding.

# Perform compression and encoding

You can use the movie file `/Assets/video.m4v` to test this app. Copy the file to your desktop or other working directory, and then open Terminal to that directory and run the following command, where `xxx` is a unique string that Xcode generates. Use autocomplete before the `xxx` component to complete the path for that directory.


```
~/Library/Developer/Xcode/DerivedData/VTEncoderForStreaming-  
xxx/Build/Products/Debug/VTEncoderForStreaming-Swift video.m4v
```

Pass the `--help` option for additional configuration options.


---

## See Also

### Compression

 Encoding video for low-latency conferencing

Configure a compression session to optimize encoding for video-conferencing apps.

 Encoding video for offline transcoding

Configure a compression session to transcode video in offline workflows.

 `VTCompressionSession`

An object that compresses video data.

 `VTDecompressionSession`

An object that decompresses video data.

 `VTFrameSilo`

An object that stores sample buffers from a multipass encoding session.

 `VTMultiPassStorage`

An object that stores video encoding metadata from a multipass encoding session.