```
import CoreFoundation
import CoreMedia
import CoreVideo
import Foundation
import VideoToolbox.VTBase
import
VideoToolbox.VTCompressionProperties
import VideoToolbox.VTCompressionSession
import
VideoToolbox.VTDecompressionProperties
import
VideoToolbox.VTDecompressionSession
import VideoToolbox.VTErrors
import VideoToolbox.VTFrameSilo
import
VideoToolbox.VTHDRPerFrameMetadataGenerat
ionSession
import VideoToolbox.VTMultiPassStorage
import
VideoToolbox.VTPixelRotationProperties
import
VideoToolbox.VTPixelRotationSession
import
VideoToolbox.VTPixelTransferProperties
import
VideoToolbox.VTPixelTransferSession
import
VideoToolbox.VTProfessionalVideoWorkflow
import
VideoToolbox.VTRAWProcessingProperties
import
VideoToolbox.VTRAWProcessingSession
import VideoToolbox.VTSession
```

```
import VideoToolbox.VTUtilities
import VideoToolbox.VTVideoEncoderList
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

/**
VTCompressionSessionEncodeMultiImageFrame

Call this function to present a multi-
image frame to the compression session.
Encoded frames may or may not be output
before the function returns.

The client should not modify the pixel
data after making this call.
The session and/or encoder will retain
the image buffers as long as necessary.
Cannot be called with a session created
with a VTCompressionOutputCallback.

- Parameters:
    - session: The compression session.
    - taggedBuffers: An array of
CMTaggedBuffer containing the multiple
images for a video frame to be
compressed.
    - presentationTimeStamp: The
presentation timestamp for this frame, to
be attached to the sample buffer.
    Each presentation timestamp passed to
a session must be greater than the
previous one.
```

- duration: The presentation duration for this frame, to be attached to the sample buffer.
If you do not have duration information, pass kCMTimeInvalid.
- frameProperties: Contains key/value pairs specifying additional properties for encoding this frame.
Note that some session properties may also be changed between frames.
Such changes have effect on subsequently encoded frames.
- infoFlagsOut: Points to a VTEncodeInfoFlags to receive information about the encode operation.
The kVTEncodeInfo_Asynchronous bit may be set if the encode is (or was) running asynchronously.
The kVTEncodeInfo_FrameDropped bit may be set if the frame was dropped (synchronously).
Pass NULL if you do not want to receive this information.
- outputHandler: The block to be called when encoding the frame is completed.
This block may be called asynchronously, on a different thread from the one that calls VTCompressionSessionEncodeMultiImageFrame.

- Returns: The OSStatus indicating the

```
result: noErr if compression was
successful; an error code if compression
was not successful.
 */
@available(macOS 14.0, iOS 17.0, visionOS
1.0, *)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
public func
VTCompressionSessionEncodeMultiImageFrame
(_ session: VTCompressionSession,
taggedBuffers: [CMTaggedBuffer],
presentationTimeStamp: CMTime, duration:
CMTime, frameProperties: CFDictionary?,
infoFlagsOut:
UnsafeMutablePointer<VTEncodeInfoFlags>?,
outputHandler: @escaping
VTCompressionOutputHandler) -> OSStatus

/**
    VTDecompressionSessionCreate

    Creates a session for decompressing
video frames.
    Decompressed frames will be emitted
through calls to an output handler
provided with each frame.
    - Parameters:
        - allocator: An allocator for the
session.  Pass NULL to use the default
allocator.
        - videoFormatDescription:
    Describes the source video frames.
```

```
        - videoDecoderSpecification:
Specifies a particular video decoder that
must be used.
            Pass NULL to let the video
toolbox choose a decoder.
        -
destinationImageBufferAttributes:
Describes requirements for emitted pixel
buffers.
            Pass NULL to set no
requirements.
        - decompressionSessionOut: Points
to the new decompression session if
successful.

    - Returns: The OSStatus indicating
the result: noErr if session creation was
successful; an error code if creation was
not successful.
*/
@available(macOS 14.0, iOS 17.0, tvOS
17.0, visionOS 1.0, *)
@available(watchOS, unavailable)
public func
VTDecompressionSessionCreate(allocator:
CFAllocator?, formatDescription
videoFormatDescription:
CMVideoFormatDescription,
decoderSpecification
videoDecoderSpecification: CFDictionary?,
imageBufferAttributes
destinationImageBufferAttributes:
CFDictionary?, decompressionSessionOut:
```

```
UnsafeMutablePointer<VTDecompressionSessi
on?>) -> OSStatus

/**
    VTDecompressionSessionDecodeFrame

    Decompresses a video frame.

    Cannot be called with a session
created with a
VTDecompressionOutputCallbackRecord.
    When you decode a frame, you provide
a closure to be called for that
decompressed frame.  This will not
necessarily be called in display order.
    If this call returns an error, the
closure will not be called.
    - Parameters:
        - session: The decompression
session.
        - sampleBuffer: A CMSampleBuffer
containing one or more video frames.
        - decodeFlags: A bitfield of
directives to the decompression session
and decoder.
            The
kVTDecodeFrame_EnableAsynchronousDecompre
ssion bit indicates whether the video
decoder
            may decompress the frame
asynchronously.
            The
kVTDecodeFrame_EnableTemporalProcessing
```

bit indicates whether the decoder may delay calls to the output callback
            so as to enable processing in temporal (display) order.
            If both flags are clear, the decompression shall complete and your output callback function will be called
            before VTDecompressionSessionDecodeFrame returns.
            If either flag is set, VTDecompressionSessionDecodeFrame may return before the output callback function is called.
        - infoFlagsOut: Points to a VTDecodeInfoFlags to receive information about the decode operation.
            The kVTDecodeInfo_Asynchronous bit may be set if the decode is (or was) running asynchronously.
            The kVTDecodeInfo_FrameDropped bit may be set if the frame was dropped (synchronously).
            Pass NULL if you do not want to receive this information.
        - completionHandler: The closure to be called when decoding the frame is completed.
            If the VTDecompressionSessionDecodeFrame call returns an error, the closure will not be called.

Only one of imageBuffer and taggedBuffers can be non-nil.
The parameters of the completionHandler are:
- status: noErr if decompression was successful; an error code if decompression was not successful.
- infoFlags: VTDecodeInfoFlags containing information about the decode operation.
The kVTDecodeInfo_Asynchronous bit may be set if the decode ran asynchronously.
The kVTDecodeInfo_FrameDropped bit may be set if the frame was dropped.
If the kVTDecodeInfo_ImageBufferModifiable bit is set, it is safe for the client to modify the imageBuffer.
- imageBuffer: Contains the decompressed frame, if decompression was successful and the
CMSampleBuffer contained a single image frame; otherwise, nil.
IMPORTANT: The video decompressor may still be referencing the imageBuffer returned in this
callback if the kVTDecodeInfo_ImageBufferModifiable flag is not set. Unless this flag
is set, it is not safe to modify the returned imageBuffer.

```
            - taggedBuffers: Contains the
decompressed frame's multiple images, if
decompression was
                successful and the
CMSampleBuffer contained a multi-image
frame; otherwise, nil.
                IMPORTANT: The video
decompressor may still be referencing the
pixelBuffers returned in this
                callback if the
kVTDecodeInfo_ImageBufferModifiable flag
is not set.  Unless this flag
                is set, it is not safe to
modify the returned pixelBuffers.
            - presentationTimeStamp: A
CMTime value for the frame's presentation
timestamp; kCMTimeInvalid if not
available.
            - presentationDuration: A
CMTime value for the frame's presentation
duration; kCMTimeInvalid if not
available.

    - Returns: The OSStatus indicating
the result: noErr if decompression was
successful; an error code if
decompression was not successful.
 */
@available(macOS 14.0, iOS 17.0, visionOS
1.0, *)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
public func
```

```
VTDecompressionSessionDecodeFrame(_
session: VTDecompressionSession,
sampleBuffer: CMSampleBuffer, flags
decodeFlags: VTDecodeFrameFlags,
infoFlagsOut:
UnsafeMutablePointer<VTDecodeInfoFlags>?,
completionHandler: @escaping @Sendable (_
status: OSStatus, _ infoFlags:
VTDecodeInfoFlags, _ imageBuffer:
CVImageBuffer?, _ taggedBuffers:
[CMTaggedBuffer]?, _
presentationTimeStamp: CMTime, _
presentationDuration: CMTime) -> Void) ->
OSStatus

/**
 */
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
@available(watchOS, unavailable)
public class
VTHDRPerFrameMetadataGenerationSession {

    public enum HDRFormat : Int, Sendable
{

        case dolbyVision

        /// Creates a new instance with
the specified raw value.
        ///
        /// If there is no value of the
type that corresponds with the specified
```

```
raw
        /// value, this initializer
returns `nil`. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
Legal
        ///     }
        ///
        ///     print(PaperSize(rawValue:
"Legal"))
        ///     // Prints
"Optional("PaperSize.Legal")"
        ///
        ///     print(PaperSize(rawValue:
"Tabloid"))
        ///     // Prints "nil"
        ///
        /// - Parameter rawValue: The raw
value to use for the new instance.
        public init?(rawValue: Int)

        /// The raw type that can be used
to represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type,
but there may be values of the `RawValue`
        /// type that don't have a
corresponding value of the conforming
```

```
type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        @available(watchOS, unavailable)
        public typealias RawValue = Int

        /// The corresponding value of
the raw type.
        ///
        /// A new instance initialized
with `rawValue` will be equivalent to
this
        /// instance. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
Legal
        ///     }
        ///
        ///     let selectedSize =
PaperSize.Letter
        ///
print(selectedSize.rawValue)
        ///     // Prints "Letter"
        ///
        ///     print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        ///     // Prints "true"
        public var rawValue: Int { get }
    }

    public init(framesPerSecond: Float,
```

```
hdrFormats:
[VTHDRPerFrameMetadataGenerationSession.H
DRFormat]? = nil) throws

    public func attachMetadata(to:
CVPixelBuffer, sceneChange: Bool = false)
throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
@available(watchOS, unavailable)
extension
VTHDRPerFrameMetadataGenerationSession.HD
RFormat : Equatable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
@available(watchOS, unavailable)
extension
VTHDRPerFrameMetadataGenerationSession.HD
RFormat : Hashable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
@available(watchOS, unavailable)
extension
VTHDRPerFrameMetadataGenerationSession.HD
RFormat : RawRepresentable {
}
```

```
@available(macOS 15.0, *)
@available(iOS, unavailable)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
@available(visionOS, unavailable)
public func
VTRAWProcessorExtensionProperties(_
formatDesc: CMFormatDescription) throws
-> [VTExtensionPropertiesKey : Any]

@available(macOS 15.0, *)
@available(iOS, unavailable)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
@available(visionOS, unavailable)
public func
VTVideoDecoderExtensionProperties(_
formatDesc: CMFormatDescription) throws
-> [VTExtensionPropertiesKey : Any]

@available(macOS 15.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
@available(iOS, unavailable)
@available(visionOS, unavailable)
extension VTRAWProcessingSession {

    /// A
VTRAWProcessingSession.Parameter
expresses a control or a set of controls
used to influence subsequent processFrame
calls on a VTRAWProcessingSession.
    /// Parameters can represent Boolean
```

```
options, Integer or Float ranges, Lists,
or subgroups.
    /// All Parameters have a collection
of Details containing a localized name
suitable for dispay in UI, a longer
localized description string, and a
boolean indicaing whether it is enabled.
    /// All Parameter.Details, except for
subgroups, must have a "key" String used
to uniquely identify that parameter
    /// All Parameters other than
subgroups have a collection of Values
containing a mandatory "initial" value,
and optional "neutral" and "camera"
values.
    /// IntegerParameter and
FloatParameter are required to have
"minimum" and "maximum" values in their
Values
    ///
    /// Parameter arrays are created and
returned by the VideoToolbox framework.
    public enum Parameter : Equatable,
Sendable {

        case
bool(VTRAWProcessingSession.Parameter.Boo
leanParameter)

        case
int(VTRAWProcessingSession.Parameter.Inte
gerParameter)
```

```
        case
float(VTRAWProcessingSession.Parameter.Fl
oatParameter)

        case
list(VTRAWProcessingSession.Parameter.Lis
tParameter)

        case subgroup(details:
VTRAWProcessingSession.Parameter.Details,
elements:
[VTRAWProcessingSession.Parameter])

        public struct Details :
Equatable, Sendable {

            public var name: String { get
}

            public var description:
String { get }

            public var isEnabled: Bool {
get }

            /// Returns a Boolean value
indicating whether two values are equal.
            ///
            /// Equality is the inverse
of inequality. For any values `a` and
`b`,
            /// `a == b` implies that
`a != b` is `false`.
```

```
            ///
            /// - Parameters:
            ///   - lhs: A value to
compare.
            ///   - rhs: Another value to
compare.
            public static func == (a:
VTRAWProcessingSession.Parameter.Details,
b:
VTRAWProcessingSession.Parameter.Details)
-> Bool
        }

        public struct Values<Value> :
Equatable, Sendable where Value :
Equatable, Value : Sendable {

            public var initial: Value {
get }

            public var current: Value {
get }

            public var neutral: Value? {
get }

            public var camera: Value? {
get }

            public var minimum: Value? {
get }

            public var maximum: Value? {
```

```
get }

            /// Returns a Boolean value
indicating whether two values are equal.
            ///
            /// Equality is the inverse
of inequality. For any values `a` and
`b`,
            /// `a == b` implies that
`a != b` is `false`.
            ///
            /// - Parameters:
            ///   - lhs: A value to
compare.
            ///   - rhs: Another value to
compare.
            public static func == (a:
VTRAWProcessingSession.Parameter.Values<V
alue>, b:
VTRAWProcessingSession.Parameter.Values<V
alue>) -> Bool
        }

        public struct BooleanParameter :
Equatable, Sendable {

            public var details:
VTRAWProcessingSession.Parameter.Details
{ get }

            public var values:
VTRAWProcessingSession.Parameter.Values<B
ool> { get }
```

```
            public var key: String {
get }

            /// Returns a Boolean value
indicating whether two values are equal.
            ///
            /// Equality is the inverse
of inequality. For any values `a` and
`b`,
            /// `a == b` implies that
`a != b` is `false`.
            ///
            /// - Parameters:
            ///   - lhs: A value to
compare.
            ///   - rhs: Another value to
compare.
            public static func == (a:
VTRAWProcessingSession.Parameter.BooleanP
arameter, b:
VTRAWProcessingSession.Parameter.BooleanP
arameter) -> Bool
        }

        public struct IntegerParameter :
Equatable, Sendable {

            public var details:
VTRAWProcessingSession.Parameter.Details
{ get }

            public var values:
```

```
VTRAWProcessingSession.Parameter.Values<I
nt> { get }

            public var key: String {
get }

            /// Returns a Boolean value
indicating whether two values are equal.
            ///
            /// Equality is the inverse
of inequality. For any values `a` and
`b`,
            /// `a == b` implies that
`a != b` is `false`.
            ///
            /// - Parameters:
            ///   - lhs: A value to
compare.
            ///   - rhs: Another value to
compare.
            public static func == (a:
VTRAWProcessingSession.Parameter.IntegerP
arameter, b:
VTRAWProcessingSession.Parameter.IntegerP
arameter) -> Bool
        }

        public struct FloatParameter :
Equatable, Sendable {

            public var details:
VTRAWProcessingSession.Parameter.Details
{ get }
```

```
            public var values:
VTRAWProcessingSession.Parameter.Values<F
loat> { get }

            public var key: String {
get }

            /// Returns a Boolean value
indicating whether two values are equal.
            ///
            /// Equality is the inverse
of inequality. For any values `a` and
`b`,
            /// `a == b` implies that
`a != b` is `false`.
            ///
            /// - Parameters:
            ///   - lhs: A value to
compare.
            ///   - rhs: Another value to
compare.
            public static func == (a:
VTRAWProcessingSession.Parameter.FloatPar
ameter, b:
VTRAWProcessingSession.Parameter.FloatPar
ameter) -> Bool
        }

        public struct ListParameter :
Equatable, Sendable {

            public struct Element :
```

```
Equatable, Sendable {

                public var id: Int {
get }

                public var details:
VTRAWProcessingSession.Parameter.Details
{ get }

                /// Returns a Boolean
value indicating whether two values are
equal.
                ///
                /// Equality is the
inverse of inequality. For any values `a`
and `b`,
                /// `a == b` implies that
`a != b` is `false`.
                ///
                /// - Parameters:
                ///   - lhs: A value to
compare.
                ///   - rhs: Another
value to compare.
                public static func == (a:
VTRAWProcessingSession.Parameter.ListPara
meter.Element, b:
VTRAWProcessingSession.Parameter.ListPara
meter.Element) -> Bool
            }

            public var details:
VTRAWProcessingSession.Parameter.Details
```

```
{ get }

            public var values:
VTRAWProcessingSession.Parameter.Values<I
nt> { get }

            public var elements:
[VTRAWProcessingSession.Parameter.ListPar
ameter.Element] { get }

            public var key: String {
get }

            /// Returns a Boolean value
indicating whether two values are equal.
            ///
            /// Equality is the inverse
of inequality. For any values `a` and
`b`,
            /// `a == b` implies that
`a != b` is `false`.
            ///
            /// - Parameters:
            ///   - lhs: A value to
compare.
            ///   - rhs: Another value to
compare.
            public static func == (a:
VTRAWProcessingSession.Parameter.ListPara
meter, b:
VTRAWProcessingSession.Parameter.ListPara
meter) -> Bool
        }
```

```
        /// Returns a Boolean value
indicating whether two values are equal.
        ///
        /// Equality is the inverse of
inequality. For any values `a` and `b`,
        /// `a == b` implies that `a !=
b` is `false`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func == (a:
VTRAWProcessingSession.Parameter, b:
VTRAWProcessingSession.Parameter) -> Bool
    }
}

@available(macOS 15.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
@available(iOS, unavailable)
@available(visionOS, unavailable)
extension VTRAWProcessingSession :
@unchecked Sendable {

    /// The array of processing
parameters available for this
VTRAWProcessingSession
    /// This call will throw an error if
the RAW Processor extension process is
unreachable
```

```
    public var processingParameters:
[VTRAWProcessingSession.Parameter] { get
throws }

    /// Returns an AsyncSequence  which
provides updates to the processing
Parameter array if the processing
extension makes changes to the set of
Parameters.
    /// These changes could be:
    ///     - adding or removing
Parameters
    ///     - enabling/disabling
Parameters
    ///     - changing default values for
a Parameter
    public func parameters() -> any
AsyncSequence<[VTRAWProcessingSession.Par
ameter], Never>

    /// Allows the client to set the
value for one or more of the processing
parameters.
    ///
    /// - Parameter values: a dictionary
where keys correspond to
Parameter.details.key for the parameters
being set, and the value is appropriate
to the Parameter type
    /// This call will throw an error if
the RAW Processor extension process is
unreachable or if the provided parameter
values are invalid or out of range
```

```
    public func updateParameter(values:
[String : Any]) throws
}

@available(macOS 15.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
@available(iOS, unavailable)
@available(visionOS, unavailable)
extension VTRAWProcessingSession {

    /// Processes an input CVPixelBuffer
    ///
    /// - Parameters
    /// frame: a CVPixelBuffer to be
processed
    /// returns a processed CVPixelBuffer
if successful or throws an Error if
unsuccessful
    /// This call will throw an error if
the RAW Processor extension process is
unreachable or the CVPixelBuffer is not
compatible with the processor
    @available(macOS 15.0, *)
    @available(watchOS, unavailable)
    @available(tvOS, unavailable)
    @available(iOS, unavailable)
    @available(visionOS, unavailable)
    public func process(frame:
CVPixelBuffer) async throws ->
CVPixelBuffer
}
```

```
@available(macOS 15.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
@available(iOS, unavailable)
@available(visionOS, unavailable)
extension VTRAWProcessingSession {

    /// this gets or sets the preferred
Metal device to be used for any Metal
based processing being performed by the
RAW Processing Extension
    /// Setting 'nil' indicates that the
client has no preferred Metal device.
    /// Getting 'nil' indicates that no
preferred device was set or that the
processor does not use Metal for frame
processing.
    @available(macOS 15.0, *)
    @available(watchOS, unavailable)
    @available(tvOS, unavailable)
    @available(iOS, unavailable)
    @available(visionOS, unavailable)
    public var metalDevice: (any
MTLDevice)?
}
```