```
import AVFoundation
import Accelerate
import Accelerate.vecLib
import Accelerate.vecLib.vDSP
import CoreFoundation
import CoreGraphics
import CoreImage
import CoreML
import CoreMedia
import CoreVideo
import Darwin
import Foundation
import ImageIO
import MachO
import
Vision.VNCalculateImageAestheticsScoresRe
quest
import Vision.VNClassifyImageRequest
import Vision.VNCoreMLRequest
import Vision.VNDefines
import
Vision.VNDetectAnimalBodyPoseRequest
import Vision.VNDetectBarcodesRequest
import Vision.VNDetectContoursRequest
import
Vision.VNDetectDocumentSegmentationReques
t
import
Vision.VNDetectFaceCaptureQualityRequest
import
Vision.VNDetectFaceLandmarksRequest
import
Vision.VNDetectFaceRectanglesRequest
```

```
import Vision.VNDetectHorizonRequest
import
Vision.VNDetectHumanBodyPose3DRequest
import
Vision.VNDetectHumanBodyPoseRequest
import
Vision.VNDetectHumanHandPoseRequest
import
Vision.VNDetectHumanRectanglesRequest
import Vision.VNDetectRectanglesRequest
import
Vision.VNDetectTextRectanglesRequest
import Vision.VNDetectTrajectoriesRequest
import Vision.VNDetectedPoint
import Vision.VNError
import Vision.VNFaceLandmarks
import Vision.VNFaceObservationAccepting
import
Vision.VNGenerateAttentionBasedSaliencyIm
ageRequest
import
Vision.VNGenerateForegroundInstanceMaskRe
quest
import
Vision.VNGenerateImageFeaturePrintRequest
import
Vision.VNGenerateObjectnessBasedSaliencyI
mageRequest
import
Vision.VNGenerateOpticalFlowRequest
import
Vision.VNGeneratePersonInstanceMaskReques
t
```

```
import
Vision.VNGeneratePersonSegmentationReques
t
import Vision.VNGeometry
import Vision.VNGeometryUtils
import
Vision.VNHumanBodyRecognizedPoint3D
import Vision.VNImageRegistrationRequest
import Vision.VNObservation
import Vision.VNRecognizeAnimalsRequest
import Vision.VNRecognizeTextRequest
import Vision.VNRecognizedPoint3D
import Vision.VNRequest
import Vision.VNRequestHandler
import Vision.VNRequestRevisionProviding
import Vision.VNStatefulRequest
import Vision.VNTargetedImageRequest
import
Vision.VNTrackHomographicImageRegistratio
nRequest
import Vision.VNTrackObjectRequest
import Vision.VNTrackOpticalFlowRequest
import Vision.VNTrackRectangleRequest
import
Vision.VNTrackTranslationalImageRegistrat
ionRequest
import Vision.VNTrackingRequest
import Vision.VNTypes
import Vision.VNUtils
import Vision.VNVideoProcessor
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims
```

```swift
import os
import simd
import simd.types

@available(macOS 10.13, *)
public var VNVisionVersionNumber: Double

/// An observation that provides the
animal body points the analysis
recognizes.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct AnimalBodyPoseObservation :
VisionObservation, PoseProviding {

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
```

```
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Retrieves a dictionary of joints
for a joint group.
    public func allJoints(in groupName:
AnimalBodyPoseObservation.PoseJointsGroup
Name? = nil) ->
[AnimalBodyPoseObservation.PoseJointName
: Joint]

    public typealias PoseJointName =
AnimalBodyPoseObservation.JointName

    public typealias PoseJointsGroupName
=
AnimalBodyPoseObservation.JointsGroupName

    public enum JointName : String,
Hashable, Sendable, Codable {

        /// A joint name that represents
the top of the left ear.
        case leftEarTop
        /// A joint name that represents
the middle of the left ear.
        case leftEarMiddle

        /// A joint name that represents
the bottom of the left ear.
        case leftEarBottom

        /// A joint name that represents
the left eye.
        case leftEye

        /// A joint name that represents
the neck.
        case neck

        /// A joint name that represents
the nose.
        case nose

        /// A joint name that represents
the right eye.
        case rightEye

        /// A joint name that represents
the top of the right ear.
        case rightEarTop

        /// A joint name that represents
the middle of the right ear.
        case rightEarMiddle
        /// A joint name that represents
the bottom of the right ear.
        case rightEarBottom

        /// A joint name that represents
the back of the left elbow.
        case leftBackElbow

        /// A joint name that represents
the front of the left elbow.
        case leftFrontElbow

        /// A joint name that represents
the front of the right elbow.
        case rightFrontElbow

        /// A joint name that represents
the back of the right elbow.
        case rightBackElbow

        /// A joint name that represents
the back of the left knee.
        case leftBackKnee

        /// A joint name that represents
the front of the left knee.
        case leftFrontKnee

        /// A joint name that represents
the back of the right knee.
        case rightBackKnee

        /// A joint name that represents
```

```
the front of the right knee.
        case rightFrontKnee

        /// A joint name that represents
the back of the left paw.
        case leftBackPaw

        /// A joint name that represents
the front of the left paw.
        case leftFrontPaw

        /// A joint name that represents
the back of the right paw.
        case rightBackPaw

        /// A joint name that represents
the front of the right paw.
        case rightFrontPaw

        /// A joint name that represents
the top of the tail.
        case tailTop

        /// A joint name that represents
the middle of the tail.
        case tailMiddle

        /// A joint name that represents
the bottom of the tail.
        case tailBottom

        /// 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: String)

        /// 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, *)
        public typealias RawValue =
String

        /// 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: String { get
}
    }

    /// The joint group names for an
animal body pose.
    public enum JointsGroupName : String,
CaseIterable, Hashable, Sendable {

        /// A group name that represents
the forelegs.
        case forelegs

        /// A group name that represents
the head.
        case head

        /// A group name that represents
the hindlegs.
        case hindlegs

        /// A group name that represents
the tail.
        case tail

        /// A group name that represents
the trunk.
        case trunk

        /// 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: String)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[AnimalBodyPoseObservation.JointsGroupNam
e]

        /// 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, *)
        public typealias RawValue =
String

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[AnimalBodyPoseObservation.JointsGroupNam
e] { get }

        /// 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: 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:
AnimalBodyPoseObservation, b:
AnimalBodyPoseObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension AnimalBodyPoseObservation :
Codable {

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
```

```
    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws
}

extension AnimalBodyPoseObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNAnimalBodyPoseObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
AnimalBodyPoseObservation.JointName :
RawRepresentable {
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
AnimalBodyPoseObservation.JointsGroupName
: RawRepresentable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct BarcodeObservation :
VisionObservation, QuadrilateralProviding
{

    public enum CompositeType : Codable,
Equatable, Hashable, Sendable {

        /// A type that represents trade
items in bulk.
        case gs1TypeA

        /// A type that represents trade
items by piece.
        case gs1TypeB

        /// A type that represents trade
items in varying quantity.
        case gs1TypeC

        /// A type that represents a
linked composite type.
        case linked

        /// 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:
BarcodeObservation.CompositeType, b:
BarcodeObservation.CompositeType) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
```

```
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
```

```
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// A string value that represents
the barcode payload.
    ///
    /// Depending on the symbology or the
payload data itself, a string
representation of the payload may not be
available.
    public let payloadString: String?

    /// The raw data representation of
the barcode’s payload.
    public let payloadData: Data?

    /// The supplemental code decoded as
a string value.
    public let supplementalPayloadString:
String?

    /// The raw data representation of
the barcode’s supplemental payload.
    public let supplementalPayloadData:
Data?

    /// The supplemental composite type.
    ///
    /// Currently, this can only refer to
the composite flag of the 2D symbology as
part of a GS1 composite symbology.
    /// This attribute only exists when
the primary descriptor is the 1D
symbology of a GS1 composite symbology,
    /// and of which a valid 2D
counterpart has been coalesced into.
    public let supplementalCompositeType:
BarcodeObservation.CompositeType?
```

```
    /// A Boolean value that indicates
whether the barcode carries any global
standards data.
    public let isGS1DataCarrier: Bool

    /// The symbology of the observed
barcode.
    public let symbology:
BarcodeSymbology

    /// A Boolean value that indicates
whether the barcode is color inverted.
    public let isColorInverted: Bool

    /// The coordinates of the upper-left
corner of the quadrilateral.
    public var topLeft: NormalizedPoint

    /// The coordinates of the upper-
right corner of the quadrilateral.
    public var topRight: NormalizedPoint

    /// The coordinates of the lower-
right corner of the quadrilateral.
    public var bottomRight:
NormalizedPoint

    /// The coordinates of the lower-left
corner of the quadrilateral.
    public var bottomLeft:
NormalizedPoint

    /// The unique identifier for the
```

```
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
```

```
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
BarcodeObservation, b:
BarcodeObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension BarcodeObservation : Codable {
    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension BarcodeObservation {
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNBarcodeObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum BarcodeSymbology :
CaseIterable, Codable, Equatable,
Hashable, Sendable {

    case aztec

    case code39

    case code39Checksum

    case code39FullASCII

    case code39FullASCIIChecksum

    case code93

    case code93i

    case code128

    case dataMatrix

    case ean8

    case ean13
    case i2of5

    case i2of5Checksum

    case itf14

    case pdf417

    case qr

    case upce

    case codabar

    case gs1DataBar

    case gs1DataBarExpanded

    case gs1DataBarLimited

    case microPDF417

    case microQR

    case msiPlessey

    /// 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:
BarcodeSymbology, b: BarcodeSymbology) ->
Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
```

```
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// A type that can represent a
collection of all values of this type.
    @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
    public typealias AllCases =
[BarcodeSymbology]

    /// A collection of all values of
this type.
    nonisolated public static var
allCases: [BarcodeSymbology] { get }

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/// A protocol for objects that have a
bounding box.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol BoundingBoxProviding {

    /// The bounding box of the object.
    ///
    /// The coordinate system is
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    var boundingBox: NormalizedRect { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
CalculateImageAestheticsScoresRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
ImageAestheticsScoresObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1
        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
CalculateImageAestheticsScoresRequest.Rev
ision, b:
CalculateImageAestheticsScoresRequest.Rev
ision) -> 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:
CalculateImageAestheticsScoresRequest.Rev
ision, b:
CalculateImageAestheticsScoresRequest.Rev
ision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
```

```
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
```

```
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
CalculateImageAestheticsScoresRequest.Rev
ision? = nil)

    public var cropAndScaleAction:
ImageCropAndScaleAction
```

```
    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
CalculateImageAestheticsScoresRequest.Rev
ision

    /// The revisions supported by
`CalculateImageAestheticsScoresRequest`.
    public static let supportedRevisions:
[CalculateImageAestheticsScoresRequest.Re
vision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
CalculateImageAestheticsScoresRequest, b:
CalculateImageAestheticsScoresRequest) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum Chirality : Codable,
Equatable, Hashable, Sendable {

    case left
    case right

    /// 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: Chirality,
b: Chirality) -> Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
```

```
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/// An object that represents
classification information that an image
analysis request produces.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct ClassificationObservation :
VisionObservation, @unchecked Sendable {
    /// The is the identifier of a
classification request. An example
classification could be a string like
'cat' or 'hotdog'.
    ///
    /// The string is defined in the
model that was used for the
classification. Usually these are
technical labels that are not localized
    /// and not meant to be used directly
to be presented to an end user in the UI.
    public let identifier: String

    /// A Boolean variable indicating
whether the observation contains
precision and recall curves.
    ///
    /// Precision refers to the
percentage of your classification results
that are relevant, while recall refers to
the percentage of total
    /// relevant results correctly
classified. If this property is true,
then you can call precision and recall-
related methods in this observation.
    /// If this property is false, then
the precision and recall-related methods
won't return meaningful data.
    public var hasPrecisionRecallCurve:
Bool { get }

    /// The unique identifier for the
observation.
```

```
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
```

```
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Determines whether the
observation has a minimum recall value
for a specific precision.
    ///
    /// - Parameters:
    ///  - minimumRecall: The minimum
desired percentage of relevant results
that the algorithm correctly classified.
    ///  - precision: The percentage of
classification results that are relevant.
    ///
    /// - Returns: A Boolean indicating
whether or not this classification
observation provides a minimum percentage
of relevant results that meet the desired
precision criterion.
    public func hasMinimumRecall(_
minimumRecall: Float, forPrecision
precision: Float) -> Bool

    /// Determines whether the
observation has a minimum precision value
for a specific recall.
    ///
    /// - Parameters:
    ///   - minimumPrecision: The minimum
percentage of classification results that
are relevant.
    ///   - recall: The percentage of
relevant results that the algorithm
correctly classified.
    ///
    /// - Returns: A Boolean indicating
whether or not this classification
observation provides a minimum percentage
of relevant results that meet the desired
recall criterion.
    public func hasMinimumPrecision(_
minimumPrecision: Float, forRecall
recall: Float) -> Bool

    /// The descriptor of the request
that produced the observation.
    public let
```

```
originatingRequestDescriptor:
RequestDescriptor?

    /// 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:
ClassificationObservation, b:
ClassificationObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
```

```
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension ClassificationObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension ClassificationObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNClassificationObservation)
}

/// A request to classify an image.
///
/// This type of request produces a
collection of `ClassificationObservation`
objects that describe an image.
/// Access the possible classifications
through the `supportedIdentifiers`
property.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct ClassifyImageRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
```

```
    public typealias Result =
[ClassificationObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
ClassifyImageRequest.Revision, b:
ClassifyImageRequest.Revision) -> 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:
ClassifyImageRequest.Revision, b:
ClassifyImageRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
```

```
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
```

```
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
ClassifyImageRequest.Revision? = nil)
```

```
    /// The classification identifiers
supported by the request.
    public var supportedIdentifiers:
[String] { get }

    /// How Vision crops and scales an
input-image.
    public var cropAndScaleAction:
ImageCropAndScaleAction

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
ClassifyImageRequest.Revision

    /// The revisions supported by
`ClassifyImageRequest`.
    public static let supportedRevisions:
[ClassifyImageRequest.Revision]
```

```
    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
ClassifyImageRequest, b:
ClassifyImageRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}
```

```
/// Types that represent the compute
stage.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum ComputeStage : Codable,
Equatable, Hashable, Sendable {

    /// A stage that represents where the
system performs the main functionality.
    case main

    /// A stage that represents where the
system performs additional analysis after
the main compute stage.
    case postProcessing

    /// 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:
ComputeStage, b: ComputeStage) -> Bool

    /// Hashes the essential components
of this value by feeding them into the
```

```
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
```

```
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/**
    An observation that provides all of
the detected contours in an image.
    Contours can be referenced as a
flattened array or as a tree of enclosing
parent contours to enclosed child
contours.
    ```
    ┌ -
- - -

┌───────────────────┐   │
    │   Λ   │
┌─────────┐ │
            / \   │   │
│ │   │
    │   /   \   │   │
│ │
          /   \   │   │ C
│ │   │
    │   │ A   │   │
└─────────┘ │
          \   /   │
│   │
    │   \   /   │ B
│
            \ /
```
```

```
└──────────────────────┘   │
    │        V

│
    └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─
─ ─ ─
    ```
    Contour A index 0, index path [0].
    Contour B index 1, index path [1].
    Contour C index 2, index path [1, 0].
*/
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct ContoursObservation :
VisionObservation, @unchecked Sendable {

    /// The total number of detected
contours.
    ///
    /// Use this value to determine the
number of indices available for calling
`contour(at:)`.
    public var contourCount: Int { get }

    /// The detected contours as a path
object in normalized coordinates.
    public var normalizedPath: CGPath {
get }

    /// An array of contours that don’t
have another contour enclosing them.
    ///
    /// This array constitutes the top of
```
```

```
the contour hierarchy. You can iterate
over each VNContour instance to determine
its children.
    public var topLevelContours:
[ContoursObservation.Contour] { get }

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
```

```
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
```

```
    public var description: String {
get }

    /// Retrieves the contour object at
the specified index, irrespective of
hierarchy.
    ///
    /// - Parameters:
    ///   - contourIndex: The index of
the contour to retrieve. Valid values are
in the range of 0 to contourCount - 1.
    /// - Returns: The contour object at
the specified index path, or `nil` if the
index path is invalid.
    public func contourAtIndex(_ index:
Int) -> ContoursObservation.Contour?

    /// Retrieves the contour object at
the specified index path.
    ///
    /// - Parameters:
    ///   - indexPath: The hierarchical
index path to the contour.
    /// - Returns: The contour object at
the specified index, or `nil` if the
index is invalid.
    public func countourAtIndexPath(_
indexPath: IndexPath) ->
ContoursObservation.Contour?

    /// 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:
ContoursObservation, b:
ContoursObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
```

```
extension ContoursObservation {

    public struct Contour : @unchecked
Sendable, Equatable, Hashable,
CustomStringConvertible {

        /// The aspect ratio of the
contour.
        ///
        /// The aspect ratio is the
original image’s width divided by its
height.
        public var aspectRatio: Float {
get }

        /// The contour object’s index
path.
        public var indexPath: IndexPath {
get }

        /// The contour object as a path
in normalized coordinates.
        public var normalizedPath: CGPath
{ get }

        /// The contour’s number of
points.
        public var pointCount: Int {
get }

        /// The contour’s array of points
in normalized coordinates.
        public var normalizedPoints:
```

```
[simd_float2] { get }

        /// An array of contours that
this contour encloses.
        public var childContours:
[ContoursObservation.Contour] { get }

        /// A textual representation of
this instance.
        ///
        /// Calling this property
directly is discouraged. Instead, convert
an
        /// instance of any type to a
string by using the `String(describing:)`
        /// initializer. This initializer
works with any type, and uses the custom
        /// `description` property for
types that conform to
        /// `CustomStringConvertible`:
        ///
        ///     struct Point:
CustomStringConvertible {
        ///         let x: Int, y: Int
        ///
        ///         var description:
String {
        ///             return "(\(x), \
(y))"
        ///         }
        ///     }
        ///
        ///     let p = Point(x: 21, y:
```

```
30)
        ///     let s =
String(describing: p)
        ///     print(s)
        ///     // Prints "(21, 30)"
        ///
        /// The conversion of `p` to a
string in the assignment to `s` uses the
        /// `Point` type's `description`
property.
        public var description: String {
get }

        /// Calculates the area enclosed
by the contour.
        ///
        /// - Parameters:
        ///   - useOrientedArea: Whether
to calculate the signed area (positive
for counterclockwise-oriented contours
and negative for clockwise-oriented
contours). If you specify false, the
returned area is always positive.
        ///
        /// Attempting to calculate the
area for a contour containing random
points, or with self-crossing edges,
produces undefined results.
        public func
calculateArea(useOrientedArea: Bool =
false) -> Double

        /// Calculate the perimeter of
```

```
the contour.
        public func calculatePerimeter()
-> Double

        /// Creates a Circle that
encloses the contour.
        public func boundingCircle() ->
NormalizedCircle

        ///  Simplifies the contour to a
polygon using a Ramer-Douglas-Peucker
algorithm.
        ///
        ///  - Parameters:
        ///     - epsilon: This parameter
defines the distance threshold the
algorithm uses. It preserves points whose
perpendicular distance to the line
segment they are on is greater than
epsilon, and removes all others.
        ///  - Returns: A simplified
polygon contour from the points of the
original contour.
        public func
polygonApproximation(epsilon: Float)
throws -> ContoursObservation.Contour

        /// 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:
ContoursObservation.Contour, b:
ContoursObservation.Contour) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
```

```
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }
    }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension ContoursObservation : Codable {
    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension ContoursObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNContoursObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum CoordinateOrigin {

    /// The origin is in the upper-left
corner of the image
    case upperLeft

    /// The origin is in the lower-left
corner of the image
    case lowerLeft

    /// 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:
CoordinateOrigin, b: CoordinateOrigin) ->
Bool
```

```
    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
```

```
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension CoordinateOrigin : Equatable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension CoordinateOrigin : Hashable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
CoreMLFeatureValueObservation :
VisionObservation {

    /// The name used in the model
description of the CoreML model that
produced this observation.
    public let featureName: String
    /// The feature result of a
``CoreMLRequest`` that outputs neither a
classification nor an image.
    ///
    /// Refer to Core ML documentation
and the model itself to learn about
proper handling of the content.
    public let featureValue:
MLSendableFeatureValue

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?
    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
CoreMLFeatureValueObservation, b:
CoreMLFeatureValueObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
```

```
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension CoreMLFeatureValueObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
```

```
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension CoreMLFeatureValueObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init?(_ observation:
VNCoreMLFeatureValueObservation)
}

/// A model container to use with an
image analysis request.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct CoreMLModelContainer :
Equatable, @unchecked Sendable, Hashable
{

    /// Creates a model container to use
with an image analysis request based on
the model you provide.
    ///
    /// - Parameters:
    ///   - model: The Core ML model on
which to base the Vision request.
```

```
    ///
    /// Initialization can fail if the
Core ML model you provide isn’t supported
in Vision, such as if the model doesn’t
accept an image as input.
    public init(model: MLModel,
featureProvider: (any MLFeatureProvider)?
= nil) throws

    /// The name of the `MLFeatureValue`
that Vision sets from the request
handler.
    ///
    /// By default, Vision uses the first
input found, but you can manually set
that input to another `featureName`
instead.
    public var inputImageFeatureName:
String

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
```

```
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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 == (lhs:
CoreMLModelContainer, rhs:
CoreMLModelContainer) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image analysis request that uses a
Core ML model to process images.
///
/// The results array of a Core ML-based
image analysis request contains a
different observation type, depending on
the kind of MLModel object you use:
/// * If the model predicts a single
feature, the model's `modelDescription`
object has a non-nil value for
`predictedFeatureName` and Vision treats
the model as a classifier. The results
are `ClassificationObservation` objects.
/// * If the model's outputs include at
least one output with a feature type of
`MLFeatureType.image`, Vision treats that
model as an image-to-image model. The
results are `PixelBufferObservation`
```

```
objects.
/// * Otherwise, Vision treats the model
as a general predictor model. The results
are `CoreMLFeatureValueObservation`
objects.
/// - Note: Vision forwards all
confidence values from Core ML models as-
is and doesn’t normalize them to [0, 1].
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct CoreMLRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result = [any
VisionObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
```

```
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
CoreMLRequest.Revision, b:
CoreMLRequest.Revision) -> 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:
CoreMLRequest.Revision, b:
CoreMLRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
```

```
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
```

```
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
```

```
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - model: The container for a
CoreML model
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(model:
CoreMLModelContainer, _ revision:
CoreMLRequest.Revision? = nil)

    /// The model to base the image
analysis request on.
    ///
    /// This object wraps a Core ML
model.
    public let modelContainer:
CoreMLModelContainer

    /// The classification identifiers
supported by the request. If the specific
```

```
configuration is not supported, `nil` is
returned.
    public var supportedIdentifiers:
[String]? { get }

    public var cropAndScaleAction:
ImageCropAndScaleAction

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    public var
supportedComputeStageDevices:
[ComputeStage : [MLComputeDevice]] {
get }

    /// The request's configured
revision.
    public let revision:
CoreMLRequest.Revision

    public static let supportedRevisions:
[CoreMLRequest.Revision]
```

```
    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
CoreMLRequest, b: CoreMLRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that detects an animal body
```

```
pose.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectAnimalBodyPoseRequest
: ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[AnimalBodyPoseObservation]

    public enum Revision : Sendable,
Equatable, Codable, Hashable, Comparable
{

        case revision1

        /// 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:
DetectAnimalBodyPoseRequest.Revision, b:
DetectAnimalBodyPoseRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectAnimalBodyPoseRequest.Revision, b:
DetectAnimalBodyPoseRequest.Revision) ->
Bool

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
```

```
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
```

```
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    public init(_ revision:
DetectAnimalBodyPoseRequest.Revision? =
nil)

    /// The joint names the request
supports.
    public var supportedJointNames:
[AnimalBodyPoseObservation.JointName] {
get }

    /// The joint group names the request
supports.
    public var supportedJointsGroupNames:
[AnimalBodyPoseObservation.JointsGroupNam
e] { get }

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
```

```
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectAnimalBodyPoseRequest.Revision

    public static let supportedRevisions:
[DetectAnimalBodyPoseRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectAnimalBodyPoseRequest, b:
DetectAnimalBodyPoseRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that detects barcodes in an
image.
///
/// This request returns an array of
BarcodeObservation objects, one for each
barcode it detects.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectBarcodesRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
```

```
    public typealias Result =
[BarcodeObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision4

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectBarcodesRequest.Revision, b:
DetectBarcodesRequest.Revision) -> 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:
DetectBarcodesRequest.Revision, b:
DetectBarcodesRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
```

```
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
```

```
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    public init(_ revision:
DetectBarcodesRequest.Revision? = nil)
```

```
    /// The collection of barcode
symbologies that can be recognized by the
request.
    public var supportedSymbologies:
[BarcodeSymbology] { get }

    /// The collection of specific
barcode symbologies to be detected.
    ///
    /// By default all symbologies will
be detected.
    public var symbologies:
[BarcodeSymbology]

    /// Allow multiple codes to be
coalesced into one.
    ///
    /// The default value for this
property is `false`.
    public var
coalescesCompositeSymbologies: Bool

    /// The collection of currently-
supported revisions for
`DetectBarcodesRequest`.
    public static let supportedRevisions:
[DetectBarcodesRequest.Revision]

    /// The request's configured
revision.
    public let revision:
DetectBarcodesRequest.Revision
```

```
    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
```

```
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectBarcodesRequest, b:
DetectBarcodesRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
```

```
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that detects the contours
of the edges of an image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectContoursRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
ContoursObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1
        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectContoursRequest.Revision, b:
DetectContoursRequest.Revision) -> 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:
DetectContoursRequest.Revision, b:
DetectContoursRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
```

```
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
DetectContoursRequest.Revision? = nil)

    /// The amount by which to adjust the
image contrast.
    ///
    /// Contour detection works best with
high-contrast images. The default value
of this property is 2.0, which doubles
```

```
the image contrast to achieve the most
accurate results.
    /// This property supports a value
range from 0.0  to 3.0.
    public var contrastAdjustment: Float

    /// The pixel value to use as a pivot
for the contrast.
    ///
    /// Numeric values range from 0.0 to
1.0. You can also specify `nil` to have
the framework automatically detect the
value according to image intensity.
    /// The default value is 0.5, which
indicates the pixel center.
    public var contrastPivot: Float?

    /// A Boolean value that indicates
whether the request detects a dark object
on a light background to aid in
detection.
    ///
    /// The default value is `true`.
    public var detectsDarkOnLight: Bool

    /// The maximum image dimension to
use for contour detection.
    ///
    /// Contour detection is
computationally intensive. To improve
performance, Vision scales the input
image down, while maintaining its aspect
ratio,
    /// such that its maximum dimension
is the value of this property. Vision
never scales the image up, so specifying
the maximum value ensures that
    /// the image processes in its
original size and not as a downscaled
version.
    /// The minimum value supported is
64. The default value is 512.
    public var maximumImageDimension: Int

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectContoursRequest.Revision

    public static let supportedRevisions:
[DetectContoursRequest.Revision]

    /// An enum that identifies the
request and request revision.
```

```
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectContoursRequest, b:
DetectContoursRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that detects rectangular
regions that contain text in the input
```

```
image.
///
/// Perform this request to detect a
document in an image. The result that the
request generates contains the four
corner points of a document’s
quadrilateral and saliency mask.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
DetectDocumentSegmentationRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
DetectedDocumentObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
```

```
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectDocumentSegmentationRequest.Revisio
n, b:
DetectDocumentSegmentationRequest.Revisio
n) -> 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:
DetectDocumentSegmentationRequest.Revisio
n, b:
DetectDocumentSegmentationRequest.Revisio
n) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)
```

```
        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
```

```
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
DetectDocumentSegmentationRequest.Revisio
n? = nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect
```

```
    /// The request's configured
revision.
    public let revision:
DetectDocumentSegmentationRequest.Revisio
n

    /// The collection of currently-
supported revisions for
`DetectDocumentSegmentationRequest`.
    public static let supportedRevisions:
[DetectDocumentSegmentationRequest.Revisi
on]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
```

```
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectDocumentSegmentationRequest, b:
DetectDocumentSegmentationRequest) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
```

```
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that determines the capture
quality of faces in a photo.
///
/// This request produces one
``FaceObservation`` object for each face
analyzed. The `FaceObservation` will have
a non-nil `captureQuality` containing a
score that ranges in value from 0 to 1.
Faces with quality closer to 1 are better
lit, sharper, and more centrally
positioned than faces with quality closer
to 0.
///
/// By default, a capture quality request
first locates all faces in the input
image, then analyzes each to detect
capture quality.
/// If you’ve already located all the
faces in an image, or want to detect
```

```
capture quality in only a subset of the
faces in the image, set the
`inputFaceObservations` property to an
array of `FaceObservation` objects
representing the faces you want to
analyze. You can either use face
observations output by a
``DetectFaceRectanglesRequest`` or
manually create `FaceObservation`
instances with the bounding boxes of the
faces you want to analyze.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
DetectFaceCaptureQualityRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[FaceObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision3

        /// Returns a Boolean value
indicating whether the value of the first
```

```
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectFaceCaptureQualityRequest.Revision,
b:
DetectFaceCaptureQualityRequest.Revision)
-> 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:
DetectFaceCaptureQualityRequest.Revision,
b:
DetectFaceCaptureQualityRequest.Revision)
-> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
```

```
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
DetectFaceCaptureQualityRequest.Revision?
= nil)

    /// An array of `FaceObservation`
objects to process as part of the
request.
    ///
    /// The default is nil. When nil,
Vision will first perform a
`DetectFaceRectanglesRequest` and process
all faces detected.
```

```
    public var inputFaceObservations:
[FaceObservation]?

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectFaceCaptureQualityRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectFaceCaptureQualityRequest`.
    public static let supportedRevisions:
[DetectFaceCaptureQualityRequest.Revision
]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
```

```
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectFaceCaptureQualityRequest, b:
DetectFaceCaptureQualityRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image analysis request that finds
facial features like eyes and mouth in an
image.
///
/// By default, a face landmarks request
first locates all faces in the input
```

```
image, then analyzes each to detect
facial features.
/// If you’ve already located all the
faces in an image, or want to detect
landmarks in only a subset of the faces
in the image, set the
inputFaceObservations property to an
array of ``FaceObservation`` objects
representing the faces you want to
analyze. You can either use face
observations output by a
``DetectFaceRectanglesRequest`` or
manually create `FaceObservation`
instances with the bounding boxes of the
faces you want to analyze.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
DetectFaceLandmarksRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[FaceObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {
        case revision3

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectFaceLandmarksRequest.Revision, b:
DetectFaceLandmarksRequest.Revision) ->
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:
DetectFaceLandmarksRequest.Revision, b:
DetectFaceLandmarksRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
```

```
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
```

```
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
DetectFaceLandmarksRequest.Revision? =
nil)

    /// An array of `FaceObservation`
objects to process as part of the
request.
    ///
    /// The default is nil. When nil,
Vision will first perform a
```

```
`DetectFaceRectanglesRequest` and process
all faces detected.
    public var inputFaceObservations:
[FaceObservation]?

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectFaceLandmarksRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectFaceLandmarksRequest`.
    public static let supportedRevisions:
[DetectFaceLandmarksRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }
```

```
    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectFaceLandmarksRequest, b:
DetectFaceLandmarksRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that finds faces within an
image.
///
/// This request returns faces as
rectangular bounding boxes with origin
```

```
and size.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectFaceRectanglesRequest
: ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[FaceObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision3

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectFaceRectanglesRequest.Revision, b:
DetectFaceRectanglesRequest.Revision) ->
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:
DetectFaceRectanglesRequest.Revision, b:
DetectFaceRectanglesRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
```

```
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
```

```
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
```

```
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
DetectFaceRectanglesRequest.Revision? =
nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectFaceRectanglesRequest.Revision
```

```
    /// The collection of currently-
supported revisions for
`DetectFaceRectanglesRequest`.
    public static let supportedRevisions:
[DetectFaceRectanglesRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
```

```
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectFaceRectanglesRequest, b:
DetectFaceRectanglesRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
```

```
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image analysis request that
determines the horizon angle in an image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectHorizonRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
HorizonObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
```

```
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectHorizonRequest.Revision, b:
DetectHorizonRequest.Revision) -> 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:
DetectHorizonRequest.Revision, b:
DetectHorizonRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
```

```
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
```

```
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
```

```
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
DetectHorizonRequest.Revision? = nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
```

```
revision.
    public let revision:
DetectHorizonRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectHorizonRequest`.
    public static let supportedRevisions:
[DetectHorizonRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
```

```
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectHorizonRequest, b:
DetectHorizonRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
```

```
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that detects a human body
pose.
///
/// The framework provides the detected
body pose as a
``HumanBodyPoseObservation``.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
DetectHumanBodyPose3DRequest :
ImageProcessingRequest, StatefulRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[HumanBodyPose3DObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {
        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectHumanBodyPose3DRequest.Revision, b:
DetectHumanBodyPose3DRequest.Revision) ->
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:
DetectHumanBodyPose3DRequest.Revision, b:
DetectHumanBodyPose3DRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
```

```
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    ///   - frameAnalysisSpacing: The
duration between analysis operations.
Increase this value to reduce the number
of frames analyzed on slower devices. By
default all frames will be analyzed.
```

```
    public init(_ revision:
DetectHumanBodyPose3DRequest.Revision? =
nil, frameAnalysisSpacing: CMTime? = nil)

    /// The joint names the request
supports.
    final public var supportedJointNames:
[HumanBodyPose3DObservation.JointName] {
get }

    /// The joint group names the request
supports.
    final public var
supportedJointsGroupNames:
[HumanBodyPose3DObservation.JointsGroupNa
me] { get }

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect

    /// The minimum number of frames that
the request has to process on before
reporting back any observation.
    ///
    /// This information is provided by
the request once initialized with its
required paramters.
    /// Video-based requests often need a
minimum number of frames before they can
report back any observation.
    /// An example would be that a
movement detection requires at least 5
frames to be detected.
    /// The `minimumLatencyFrameCount`
for that request would report 5 and only
after 5 frames have been processed an
observation would be returned in the
results.
    /// This latency is indicative of how
responsive a request is in respect to the
incoming data.
    final public var
minimumLatencyFrameCount: Int { get }

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime
```

```
    /// The request's configured
revision.
    final public let revision:
DetectHumanBodyPose3DRequest.Revision

    public static let supportedRevisions:
[DetectHumanBodyPose3DRequest.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}
```

```
/// A request that detects a human body
pose.
///
/// The framework provides the detected
body pose as a
``HumanBodyPoseObservation``.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
DetectHumanBodyPoseRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[HumanBodyPoseObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
```

```
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectHumanBodyPoseRequest.Revision, b:
DetectHumanBodyPoseRequest.Revision) ->
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:
DetectHumanBodyPoseRequest.Revision, b:
DetectHumanBodyPoseRequest.Revision) ->
Bool

        /// Hashes the essential
```

```
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
```

```
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    public init(_ revision:
DetectHumanBodyPoseRequest.Revision? =
nil)

    /// The joint names the request
supports.
    public var supportedJointNames:
[HumanBodyPoseObservation.JointName] {
get }

    /// The joint group names the request
supports.
    public var supportedJointsGroupNames:
[HumanBodyPoseObservation.JointsGroupName
] { get }
```

```
    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectHumanBodyPoseRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectHumanBodyPoseRequest`.
    public static let supportedRevisions:
[DetectHumanBodyPoseRequest.Revision]

    /// Detect hands of the body in
returned results if present. Requires
`.revision2`
    ///
    /// The default value for this
property is `true`.
    public var detectsHands: Bool

    /// An enum that identifies the
```

```
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectHumanBodyPoseRequest, b:
DetectHumanBodyPoseRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that detects a human hand
```

```
pose.
///
/// The framework provides the detected
hand pose as a
``HumanHandPoseObservation``.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
DetectHumanHandPoseRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[HumanHandPoseObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
```

```
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectHumanHandPoseRequest.Revision, b:
DetectHumanHandPoseRequest.Revision) ->
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:
DetectHumanHandPoseRequest.Revision, b:
DetectHumanHandPoseRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
```

```
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
```

```
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
```

```
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    public init(_ revision:
DetectHumanHandPoseRequest.Revision? =
nil)

    /// The joint names the request
supports.
    public var supportedJointNames:
[HumanHandPoseObservation.JointName] {
get }

    /// The joint group names the request
supports.
    public var supportedJointsGroupNames:
[HumanHandPoseObservation.JointsGroupName
] { get throws }
```

```
    /// The maximum number of hands to
detect in an image.
    ///
    /// The request orders detected hands
by relative size, with only the largest
ones having key points determined.
    /// The default value is 2.
    public var maximumHandCount: Int

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectHumanHandPoseRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectHumanHandPoseRequest`.
    public static let supportedRevisions:
[DetectHumanHandPoseRequest.Revision]

    /// An enum that identifies the
```

```
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectHumanHandPoseRequest, b:
DetectHumanHandPoseRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that finds rectangular
```

```
regions that contain people in an image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
DetectHumanRectanglesRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[HumanObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
```

```
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectHumanRectanglesRequest.Revision, b:
DetectHumanRectanglesRequest.Revision) ->
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:
DetectHumanRectanglesRequest.Revision, b:
DetectHumanRectanglesRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
```

```
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
```

```
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
```

```
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
DetectHumanRectanglesRequest.Revision? =
nil)

    /// A Boolean value that indicates
whether the request requires detecting a
full body or upper body only to produce a
result.
    ///
    /// The default value of `true`
indicates that the request requires
detecting a person’s upper body only to
find the bound box around it.
    public var upperBodyOnly: Bool

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
```

```
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectHumanRectanglesRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectHumanRectanglesRequest`.
    public static let supportedRevisions:
[DetectHumanRectanglesRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
```

```
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectHumanRectanglesRequest, b:
DetectHumanRectanglesRequest) -> Bool
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image analysis request that finds
projected rectangular regions in an
image.
///
/// A rectangle detection request locates
regions of an image with rectangular
shape, like credit cards, business cards,
documents, and signs.
/// The request returns its observations
in the form of `RectangleObservation`
objects, which contain normalized
coordinates of bounding boxes containing
the rectangle.
/// Use this type of request to find the
bounding boxes of rectangles in an image.
```

```
Vision returns observations for
rectangles found in all orientations and
sizes,
/// along with a confidence level to
indicate how likely it’s that the
observation contains an actual rectangle.
/// To further configure or restrict the
types of rectangles found, set properties
on the request specifying a range of
aspect ratios, sizes, and quadrature
tolerance.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectRectanglesRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[RectangleObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
```

```
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectRectanglesRequest.Revision, b:
DetectRectanglesRequest.Revision) -> 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:
DetectRectanglesRequest.Revision, b:
DetectRectanglesRequest.Revision) -> Bool
```

```
        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
```

```
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }
        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
DetectRectanglesRequest.Revision? = nil)

    /// The minimum aspect ratio of the
rectangle(s) to detect.
    ///
    /// The value should range from 0.0
to 1.0, inclusive. The default value is
0.5.
    public var minimumAspectRatio: Float

    /// The maximum aspect ratio of the
rectangle(s) to detect.
```

```
    ///
    /// The value should range from 0.0
to 1.0, inclusive. The default value is
1.0.
    public var maximumAspectRatio: Float

    /// The maximum number of degrees a
rectangle corner angle can deviate from
90°.
    ///
    /// The tolerance value should range
from 0 to 45, inclusive. The default
tolerance is 30.
    public var
quadratureToleranceDegrees: Float

    /// The minimum size of the rectangle
to be detected, as a proportion of the
smallest dimension.
    ///
    /// The value should range from 0.0
to 1.0 inclusive. The default minimum
size is 0.2.
    /// Any smaller rectangles that
Vision may have detected aren't returned.
    public var minimumSize: Float

    /// The minimum acceptable confidence
level for detected rectangles.
    ///
    /// Vision won't return rectangles
with a confidence score lower than the
specified minimum.
```

```
    /// The confidence score ranges from
0.0 to 1.0, inclusive, where 0.0
represents no confidence, and 1.0
represents full confidence. The default
minimum confidence is 0.0.
    public var minimumConfidence: Float

    /// The maximum number of rectangles
to be returned.
    ///
    /// The default is 1. Setting this
property to 0 will allow a potentially
unlimited number of observations to be
returned.
    public var maximumObservations: Int

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectRectanglesRequest.Revision
```

```
    /// The collection of currently-
supported revisions for
`DetectRectanglesRequest`.
    public static let supportedRevisions:
[DetectRectanglesRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
```

```
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectRectanglesRequest, b:
DetectRectanglesRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
```

```
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image analysis request that finds
regions of visible text in an image.
///
/// This request returns detected text
characters as rectangular bounding boxes
with origin and size.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectTextRectanglesRequest
: ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[TextObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
```

```
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
DetectTextRectanglesRequest.Revision, b:
DetectTextRectanglesRequest.Revision) ->
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:
```

```
DetectTextRectanglesRequest.Revision, b:
DetectTextRectanglesRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
```

```
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
```

```
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
DetectTextRectanglesRequest.Revision? =
nil)

    /// A Boolean value that indicates
whether the request detects character
bounding boxes.
    ///
    /// Set the value to `true` to have
the detector return character bounding
```

```
boxes as an array of
``RectangleObservation`` objects.
    public var reportCharacterBoxes: Bool

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
DetectTextRectanglesRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectTextRectanglesRequest`.
    public static let supportedRevisions:
[DetectTextRectanglesRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
```

```
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
DetectTextRectanglesRequest, b:
DetectTextRectanglesRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image analysis request that
determines the horizon angle in an image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
DetectTrajectoriesRequest :
```

```
ImageProcessingRequest, StatefulRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[TrajectoryObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
```

```
        public static func < (a:
DetectTrajectoriesRequest.Revision, b:
DetectTrajectoriesRequest.Revision) ->
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:
DetectTrajectoriesRequest.Revision, b:
DetectTrajectoriesRequest.Revision) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
```

```
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
```

```
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
```

```
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - trajectoryLength: The number
of points required to analyze to
determine that a shape follows a
parabolic path. This argument value must
be at least 5.
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    ///   - frameAnalysisSpacing: The
duration between analysis operations.
Increase this value to reduce the number
of frames analyzed on slower devices. By
default all frames will be analyzed.
    public init(trajectoryLength: Int, _
revision:
DetectTrajectoriesRequest.Revision? =
nil, frameAnalysisSpacing: CMTime? = nil)

    /// The requested target frame time
for processing trajectory detection.
    ///
    /// Use this property value for real-
time processing of frames, which requires
execution within a specific amount of
time. The request evaluates from frame-
to-frame. If processing takes longer than
the targeted time for the current frame,
it attempts to decrease the overall time
```

```
by reducing the accuracy (down to a set
minimum) for the next frame. If a frame
takes less time than the targeted time,
the request increases the accuracy (up to
a set maximum) of the next frame.
    /// The default value is indefinite,
which indicates that accuracy stays at
the predefined maximum.
    final public var targetFrameTime:
CMTime

    /// The minimum radius of the
bounding circle of the object to track.
    final public var
objectMinimumNormalizedRadius: Float

    /// The maximum radius of the
bounding circle of the object to track.
    final public var
objectMaximumNormalizedRadius: Float

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect
```

```
    /// The number of points to detect
before calculating a trajectory.
    final public let trajectoryLength:
Int

    /// The minimum number of frames that
the request has to process on before
reporting back any observation.
    ///
    /// This information is provided by
the request once initialized with its
required paramters.
    /// Video-based requests often need a
minimum number of frames before they can
report back any observation.
    /// An example would be that a
movement detection requires at least 5
frames to be detected.
    /// The `minimumLatencyFrameCount`
for that request would report 5 and only
after 5 frames have been processed an
observation would be returned in the
results.
    /// This latency is indicative of how
responsive a request is in respect to the
incoming data.
    final public var
minimumLatencyFrameCount: Int { get }

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
```

```
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime

    final public let revision:
DetectTrajectoriesRequest.Revision

    /// The collection of currently-
supported revisions for
`DetectTrajectoriesRequest `.
    public static let supportedRevisions:
[DetectTrajectoriesRequest.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
```

```
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}

/// An observation that contains a
detected document.
///
/// The observation includes the four
corner points of a document’s
quadrilateral and saliency mask.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectedDocumentObservation
: VisionObservation,
QuadrilateralProviding {

    /// A pixel buffer representing a
segmentation mask for the detected
document.
    public var globalSegmentationMask:
PixelBufferObservation

    /// The coordinates of the upper-left
corner of the quadrilateral.
    public var topLeft: NormalizedPoint

    /// The coordinates of the upper-
```

```
right corner of the quadrilateral.
    public var topRight: NormalizedPoint

    /// The coordinates of the lower-
right corner of the quadrilateral.
    public var bottomRight:
NormalizedPoint

    /// The coordinates of the lower-left
corner of the quadrilateral.
    public var bottomLeft:
NormalizedPoint

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
```

```
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
```

```
property.
    public var description: String {
get }

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// 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:
DetectedDocumentObservation, b:
DetectedDocumentObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension DetectedDocumentObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
```

```
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension DetectedDocumentObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init?(_ observation:
VNRectangleObservation)
}

/// An observation that provides the
position and extent of an image feature
that an image analysis request detects.
///
/// This class is the observation type
that ``TrackObjectRequest`` generates. It
represents an object that the Vision
request detects and tracks.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct DetectedObjectObservation :
VisionObservation, BoundingBoxProviding {
    public init(boundingBox:
NormalizedRect)

    /// The bounding box of the object.
    ///
    /// The coordinate system is
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    public var boundingBox:
NormalizedRect

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation's start time
```

```
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
DetectedObjectObservation, b:
DetectedObjectObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension DetectedObjectObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
```

```
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension DetectedObjectObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNDetectedObjectObservation)
}

public typealias DetectorKey = String

/// An enumeration of the type of element
in feature print data.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum ElementType : Codable,
Equatable, Hashable, Sendable {

    /// The elements are floating-point
numbers.
    case float
    /// The elements are double-precision
floating-point numbers.
    case double

    /// 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:
ElementType, b: ElementType) -> Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
```

```
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct FaceObservation :
VisionObservation, BoundingBoxProviding {
    /// - Parameters:
    ///     - boundingBox: The bounding
rectangle of the detected face.
    ///     - revision: The revision of
the `DetectFaceRectanglesRequest` that
provided the bounding box.
    ///
    ///     If no revision is provided, a
default value is assumed. Different
revisions can affect how the bounding box
is interpreted by Vision.
    public init(boundingBox:
NormalizedRect, revision:
DetectFaceRectanglesRequest.Revision? =
nil)

    /// The facial features of the
detected face.
    ///
    /// This value is nil for face
observations produced by a
`DetectFaceRectanglesRequest` analysis.
    /// Use the
`DetectFaceLandmarksRequest` class to
find facial features.
    public let landmarks:
FaceObservation.Landmarks2D?

    /// The roll angle of a face.
    ///
    /// This value indicates the
rotational angle of the face around the
z-axis.
```

```
    public let roll:
Measurement<UnitAngle>

    /// The yaw angle of a face.
    ///
    /// This value indicates the
rotational angle of the face around the
y-axis.
    public let yaw:
Measurement<UnitAngle>

    /// The pitch angle of a face.
    ///
    /// This value indicates the
rotational angle of the face around the
x-axis.
    public let pitch:
Measurement<UnitAngle>

    /// The quality of the face capture.
    ///
    /// This value is nil for face
observations produced by a
`DetectFaceRectanglesRequest` analysis.
    /// Use
``DetectFaceCaptureQualityRequest`` to
detect capture quality.
    public let captureQuality:
FaceObservation.CaptureQuality?

    /// The bounding box of the object.
    ///
    /// The coordinate system is
```

```
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    public var boundingBox:
NormalizedRect

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
```

```
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
```

```
    public var description: 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:
FaceObservation, b: FaceObservation) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
```

```
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// CaptureQuality
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension FaceObservation {

    /// An indicator of the quality of a
face capture.
    public struct CaptureQuality :
Sendable, Hashable, Equatable, Codable,
CustomStringConvertible {

        /// A value that indicates the
quality of the face capture.
        ///
        /// The score allows you to
compare the quality of the face in terms
of its capture attributes: lighting,
        /// blur, and prime positioning.
Use this value to compare the capture
quality of a face against other captures
of the same
        /// face in a specified set. The
value of this property value ranges from
`0.0` to `1.0`. Faces with quality closer
to `1.0`
        /// are better lit, sharper, and
more centrally positioned than faces with
quality closer to `0.0`.
        public let score: Float
        /// The descriptor of the request
that produced the capture quality.
        public let
originatingRequestDescriptor:
RequestDescriptor?

        /// A textual representation of
this instance.
        ///
        /// Calling this property
directly is discouraged. Instead, convert
an
        /// instance of any type to a
string by using the `String(describing:)`
        /// initializer. This initializer
works with any type, and uses the custom
        /// `description` property for
types that conform to
        /// `CustomStringConvertible`:
        ///
        ///     struct Point:
CustomStringConvertible {
        ///         let x: Int, y: Int
        ///
        ///         var description:
String {
        ///             return "(\(x), \
(y))"
        ///         }
        ///     }
        ///
        ///     let p = Point(x: 21, y:
```

```
30)
        ///     let s =
String(describing: p)
        ///     print(s)
        ///     // Prints "(21, 30)"
        ///
        /// The conversion of `p` to a
string in the assignment to `s` uses the
        /// `Point` type's `description`
property.
        public var description: String {
get }

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
```

```
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// 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:
FaceObservation.CaptureQuality, b:
FaceObservation.CaptureQuality) -> Bool

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
```

```
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }
}

/// Landmarks2D
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension FaceObservation {

    /// A collection of facial features
that a request detects.
    ///
    /// This class represents the set of
all detectable 2D face landmarks and
regions, exposed as properties. The
coordinates of the face landmarks are
normalized to the dimensions of the face
observation’s `boundingBox`, with the
origin at the bounding box’s lower-left
corner.
    public struct Landmarks2D : Codable,
Equatable, Sendable,
CustomStringConvertible, Hashable {

        ///
        public var allPoints:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that trace the face contour from the left
cheek, over the chin, to the right cheek.
        public var faceContour:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that outline the left eye.
        public var leftEye:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that outline the right eye.
        public var rightEye:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that trace the left eyebrow.
        public var leftEyebrow:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that trace the right eyebrow.
        public var rightEyebrow:
FaceObservation.Landmarks2D.Region {
```

```
get }

        /// The region containing points
that outline the nose.
        public var nose:
FaceObservation.Landmarks2D.Region {
get }

        ///  The region containing points
that trace the center crest of the nose.
        public var noseCrest:
FaceObservation.Landmarks2D.Region {
get }

        ///  The region containing points
that trace a vertical line down the
center of the face.
        public var medianLine:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that outline the outside of the lips.
        public var outerLips:
FaceObservation.Landmarks2D.Region {
get }

        /// The region containing points
that outline the space between the lips.
        public var innerLips:
FaceObservation.Landmarks2D.Region {
get }
```

```
        ///  The region containing the
point where the left pupil is located.
        public var leftPupil:
FaceObservation.Landmarks2D.Region {
get }

        ///  The region containing the
point where the right pupil is located.
        public var rightPupil:
FaceObservation.Landmarks2D.Region {
get }

        /// A textual representation of
this instance.
        ///
        /// Calling this property
directly is discouraged. Instead, convert
an
        /// instance of any type to a
string by using the `String(describing:)`
        /// initializer. This initializer
works with any type, and uses the custom
        /// `description` property for
types that conform to
        /// `CustomStringConvertible`:
        ///
        ///     struct Point:
CustomStringConvertible {
        ///         let x: Int, y: Int
        ///
        ///         var description:
String {
        ///             return "(\(x), \
(y))"
        ///         }
        ///     }
        ///
        ///     let p = Point(x: 21, y:
30)
        ///     let s =
String(describing: p)
        ///     print(s)
        ///     // Prints "(21, 30)"
        ///
        /// The conversion of `p` to a
string in the assignment to `s` uses the
        /// `Point` type's `description`
property.
        public var description: String {
get }

        public let
originatingRequestDescriptor:
RequestDescriptor?

        /// 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:
FaceObservation.Landmarks2D, b:
FaceObservation.Landmarks2D) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
```

```
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension FaceObservation : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
```

```
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension FaceObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNFaceObservation)
}

/// Landmarks2D.Region
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
```

```
extension FaceObservation.Landmarks2D {

    /// 2D geometry information for a
specific facial feature.
    public struct Region : Codable,
Equatable, Sendable, Hashable,
CustomStringConvertible {

        /// The set of classifications
that describe how to interpret the points
the region provides.
        public enum
PointsClassification : Codable, Hashable,
Sendable, Equatable {

            case closedPath

            case disconnected

            case openPath

            /// Hashes the essential
components of this value by feeding them
into the
            /// given hasher.
            ///
            /// Implement this method to
conform to the `Hashable` protocol. The
            /// components used for
hashing must be the same as the
components compared
            /// in your type's `==`
operator implementation. Call
```

```
`hasher.combine(_:)`
            /// with each of these
components.
            ///
            /// - Important: In your
implementation of `hash(into:)`,
            ///   don't call `finalize()`
on the `hasher` instance provided,
            ///   or replace it with a
different instance.
            ///   Doing so may become a
compile-time error in the future.
            ///
            /// - Parameter hasher: The
hasher to use when combining the
components
            ///   of this instance.
            public func hash(into hasher:
inout Hasher)

            /// 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:
FaceObservation.Landmarks2D.Region.Points
Classification, b:
FaceObservation.Landmarks2D.Region.Points
Classification) -> Bool

            /// Encodes this value into
the given encoder.
            ///
            /// If the value fails to
encode anything, `encoder` will encode an
empty
            /// keyed container in its
place.
            ///
            /// This function throws an
error if any values are invalid for the
given
            /// encoder's format.
            ///
            /// - Parameter encoder: The
encoder to write data to.
            public func encode(to
encoder: any Encoder) throws

            /// The hash value.
            ///
            /// Hash values are not
guaranteed to be equal across different
executions of
            /// your program. Do not save
hash values to use during a future
```

```
execution.
            ///
            /// - Important: `hashValue`
is deprecated as a `Hashable`
requirement. To
            ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
            ///   The compiler provides
an implementation for `hashValue` for
you.
            public var hashValue: Int {
get }

            /// Creates a new instance by
decoding from the given decoder.
            ///
            /// This initializer throws
an error if reading from the decoder
fails, or
            /// if the data read is
corrupted or otherwise invalid.
            ///
            /// - Parameter decoder: The
decoder to read data from.
            public init(from decoder: any
Decoder) throws
        }

        /// An enumeration that describes
how to interpret the points the region
provides.
        public let pointsClassification:
```

```
FaceObservation.Landmarks2D.Region.Points
Classification

        /// The array of landmark points
normalized to the bounding box of the
`FaceObservation`.
        public let points:
[NormalizedPoint]

        /// An array of precision
estimates for each landmark point.
        public let
precisionEstimatesPerPoint: [Float]?

        /// Returns the landmark points
in an image's coordinate space.
        ///
        /// - Parameters:
        ///   - imageSize: The size of
the image
        ///   - origin:
        public func
pointsInImageCoordinates(_ imageSize:
CGSize, origin: CoordinateOrigin
= .lowerLeft) -> [CGPoint]

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
```

```
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// A textual representation of
this instance.
        ///
        /// Calling this property
directly is discouraged. Instead, convert
an
        /// instance of any type to a
string by using the `String(describing:)`
        /// initializer. This initializer
```

```
works with any type, and uses the custom
        /// `description` property for
types that conform to
        /// `CustomStringConvertible`:
        ///
        ///     struct Point:
CustomStringConvertible {
        ///         let x: Int, y: Int
        ///
        ///         var description:
String {
        ///             return "(\(x), \
(y))"
        ///         }
        ///     }
        ///
        ///     let p = Point(x: 21, y:
30)
        ///     let s =
String(describing: p)
        ///     print(s)
        ///     // Prints "(21, 30)"
        ///
        /// The conversion of `p` to a
string in the assignment to `s` uses the
        /// `Point` type's `description`
property.
        public var description: String {
get }

        public let
originatingRequestDescriptor:
RequestDescriptor?
```

```
        /// 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:
FaceObservation.Landmarks2D.Region, b:
FaceObservation.Landmarks2D.Region) ->
Bool

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws
```

```
        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
```

```
    }
}

/// An observation that provides the
recognized feature print.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct FeaturePrintObservation :
VisionObservation {

    /// The feature print data.
    ///
    /// The data is divided into separate
elements. Determine the type of element
using ``elementType``, and the number of
elements
    /// using ``elementCount``.
    public let data: Data

    /// The total number of elements in
the data.
    public let elementCount: Int

    /// The type of each element in the
data.
    public let elementType: ElementType

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
```

```
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
```

```
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Computes the distance between two
feature print observations.
    ///
    /// Shorter distances indicate
greater similarity between feature
prints.
    ///
    /// - Parameters:
    ///   - featurePrint: The feature
```

```
print object to calculate the distance
to.
    public func distance(to featurePrint:
FeaturePrintObservation) throws -> Double

    /// 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:
FeaturePrintObservation, b:
FeaturePrintObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
```

```
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension FeaturePrintObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
```

```
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension FeaturePrintObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNFeaturePrintObservation)
}

/// An object that produces a heat map
that identifies the parts of an image
most likely to draw attention.
///
/// The resulting observation,
``SaliencyImageObservation``, encodes
this data as a heat map, which you can
use to highlight regions of interest.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
GenerateAttentionBasedSaliencyImageReques
t : ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
SaliencyImageObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
GenerateAttentionBasedSaliencyImageReques
t.Revision, b:
GenerateAttentionBasedSaliencyImageReques
t.Revision) -> 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:
GenerateAttentionBasedSaliencyImageReques
t.Revision, b:
GenerateAttentionBasedSaliencyImageReques
t.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
```

```
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws
```

```
        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }
    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
GenerateAttentionBasedSaliencyImageReques
t.Revision? = nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
GenerateAttentionBasedSaliencyImageReques
t.Revision

    /// The collection of currently-
supported revisions for
`GenerateAttentionBasedSaliencyImageReque
st`.
    public static let supportedRevisions:
[GenerateAttentionBasedSaliencyImageReque
st.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)
```

```
    /// 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:
GenerateAttentionBasedSaliencyImageReques
t, b:
GenerateAttentionBasedSaliencyImageReques
t) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
```

```
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that generates an instance
mask of noticeable objects to separate
from the background.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
GenerateForegroundInstanceMaskRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
InstanceMaskObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
```

```
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
GenerateForegroundInstanceMaskRequest.Rev
ision, b:
GenerateForegroundInstanceMaskRequest.Rev
ision) -> 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:
GenerateForegroundInstanceMaskRequest.Rev
ision, b:
GenerateForegroundInstanceMaskRequest.Rev
ision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)
```

```
        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
```

```
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
GenerateForegroundInstanceMaskRequest.Rev
ision? = nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
```

```
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
GenerateForegroundInstanceMaskRequest.Rev
ision

    /// The collection of currently-
supported revisions for
`GenerateForegroundInstanceMaskRequest`.
    public static let supportedRevisions:
[GenerateForegroundInstanceMaskRequest.Re
vision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
GenerateForegroundInstanceMaskRequest, b:
GenerateForegroundInstanceMaskRequest) ->
Bool
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image-based request to generate
feature prints from an image.
///
/// This request returns the feature
print data it generates as an array of
``FeaturePrintObservation objects``.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
GenerateImageFeaturePrintRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
FeaturePrintObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
GenerateImageFeaturePrintRequest.Revision
, b:
GenerateImageFeaturePrintRequest.Revision
) -> 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:
GenerateImageFeaturePrintRequest.Revision
, b:
GenerateImageFeaturePrintRequest.Revision
) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
```

```
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws
```

```
        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }
    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
GenerateImageFeaturePrintRequest.Revision
? = nil)

    /// An optional setting that tells
the algorithm how to scale an input image
before generating the feature print.
    ///
    /// Scaling is applied before
generating the feature print. The default
value is
``ImageCropAndScaleAction.scaleToFill``.
    public var cropAndScaleAction:
ImageCropAndScaleAction

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
```

```
revision.
    public let revision:
GenerateImageFeaturePrintRequest.Revision

    public static let supportedRevisions:
[GenerateImageFeaturePrintRequest.Revisio
n]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
```

```
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
GenerateImageFeaturePrintRequest, b:
GenerateImageFeaturePrintRequest) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
```

```
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A request that generates a heat map
that identifies the parts of an image
most likely to represent objects.
///
/// The resulting observation,
``SaliencyImageObservation``, encodes
this data as a heat map, which you can
use to highlight regions of interest.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
GenerateObjectnessBasedSaliencyImageReque
st : ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
SaliencyImageObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {
        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
GenerateObjectnessBasedSaliencyImageReque
st.Revision, b:
GenerateObjectnessBasedSaliencyImageReque
st.Revision) -> 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:
GenerateObjectnessBasedSaliencyImageReque
st.Revision, b:
GenerateObjectnessBasedSaliencyImageReque
st.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
```

```
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
```

```
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
GenerateObjectnessBasedSaliencyImageReque
st.Revision? = nil)
```

```
    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
GenerateObjectnessBasedSaliencyImageReque
st.Revision

    /// The collection of currently-
supported revisions for
`GenerateObjectnessBasedSaliencyImageRequ
est`.
    public static let supportedRevisions:
[GenerateObjectnessBasedSaliencyImageRequ
est.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
```

```
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
GenerateObjectnessBasedSaliencyImageReque
st, b:
GenerateObjectnessBasedSaliencyImageReque
st) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// Generates an instance mask of
individual people found in the image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
```

```
public struct
GeneratePersonInstanceMaskRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
InstanceMaskObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
GeneratePersonInstanceMaskRequest.Revisio
n, b:
GeneratePersonInstanceMaskRequest.Revisio
n) -> 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:
GeneratePersonInstanceMaskRequest.Revisio
n, b:
GeneratePersonInstanceMaskRequest.Revisio
n) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
```

```
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
```

```
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
```

```
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
GeneratePersonInstanceMaskRequest.Revisio
n? = nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
GeneratePersonInstanceMaskRequest.Revisio
n

    public static let supportedRevisions:
[GeneratePersonInstanceMaskRequest.Revisi
on]
```

```
    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)
```

```
    /// 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:
GeneratePersonInstanceMaskRequest, b:
GeneratePersonInstanceMaskRequest) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
```

```
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
GeneratePersonSegmentationRequest :
ImageProcessingRequest, StatefulRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
PixelBufferObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
```

```
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
GeneratePersonSegmentationRequest.Revisio
n, b:
GeneratePersonSegmentationRequest.Revisio
n) -> 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:
GeneratePersonSegmentationRequest.Revisio
n, b:
GeneratePersonSegmentationRequest.Revisio
n) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
```

```
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
```

```
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
GeneratePersonSegmentationRequest.Revisio
n? = nil, frameAnalysisSpacing: CMTime? =
nil)

    public enum QualityLevel :
CaseIterable, Sendable, Equatable,
Codable, Hashable {

        case accurate

        case balanced

        case fast

        /// 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:
GeneratePersonSegmentationRequest.Quality
Level, b:
GeneratePersonSegmentationRequest.Quality
Level) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
```

```
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[GeneratePersonSegmentationRequest.Qualit
yLevel]

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[GeneratePersonSegmentationRequest.Qualit
yLevel] { get }

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
```

```
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
```

```
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// Segmentation speed vs. accuracy
control.  The default is  `.accurate`.
    final public var qualityLevel:
GeneratePersonSegmentationRequest.Quality
Level

    /// The desired pixel format type of
the observation.
    final public var
outputPixelFormatType: OSType

    /// The collection of supported pixel
format types.
    final public var
supportedOutputPixelFormats: [OSType] {
get }

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
```

```
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    final public let revision:
GeneratePersonSegmentationRequest.Revisio
n

    /// The collection of currently-
supported revisions for
`GeneratePersonSegmentationRequest `.
    public static let supportedRevisions:
[GeneratePersonSegmentationRequest.Revisi
on]

    /// An enum that identifies the
```

```
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}

/// The horizon angle information that an
image analysis request detects.
///
/// Instances of this class result from
invoking a ``DetectHorizonRequest``, and
report the angle and transform of the
horizon in an image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct HorizonObservation :
```

```
VisionObservation {

    /// The transform to apply to the
detected horizon.
    ///
    /// Apply the transform's inverse to
orient the image in an upright position
and make the detected horizon level.
    public let transform:
CGAffineTransform

    /// The angle of the observed
horizon.
    ///
    /// Use the angle to orient the image
in an upright position and make the
detected horizon level.
    public let angle:
Measurement<UnitAngle>

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float
    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Creates an affine transform for
the specified image width and height.
    ///
    /// - Parameters:
    ///    - imageSize: The size of the
image.
    /// - Returns: An affine transform.
Apply the transform's inverse to orient
the image in an upright position and make
the detected horizon level.
    public func transform(for imageSize:
CGSize) -> CGAffineTransform

    /// 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:
HorizonObservation, b:
HorizonObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
```

```
extension HorizonObservation : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension HorizonObservation {
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNHorizonObservation)
}

/// An observation that provides the
three-dimensional body points the request
recognizes.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
HumanBodyPose3DObservation :
VisionObservation {

    /// Constants that identify body
height estimation techniques.
    public enum EstimationTechnique :
Sendable, Equatable, Codable, Hashable {

        /// A technique that uses a
reference height.
        case reference

        /// A technique that uses LiDAR
depth data to measure body height, in
meters.
        case measured

        /// 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:
HumanBodyPose3DObservation.EstimationTech
nique, b:
HumanBodyPose3DObservation.EstimationTech
nique) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
```

```
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
```

```
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// The technique the framework uses
to estimate body height.
    public let heightEstimationTechnique:
```

```
HumanBodyPose3DObservation.EstimationTech
nique

    /// The estimated human body height.
    public let bodyHeight:
Measurement<UnitLength>

    /// A transform from the skeleton hip
to the camera.
    public let cameraOriginMatrix:
simd_float4x4

    /// The names of the available joints
in the observation.
    public var availableJointNames:
[HumanBodyPose3DObservation.JointName] {
get }

    /// The names of the available joint
groupings in the observation.
    public var availableJointsGroupNames:
[HumanBodyPose3DObservation.JointsGroupNa
me] { get }

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
```

```
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Retrieves a  joint for a given
joint name.
    public func joint(for jointName:
HumanBodyPose3DObservation.JointName) ->
Joint3D?

    /// Retrieves a dictionary of joints
for a joint group.
    public func allJoints(in groupName:
HumanBodyPose3DObservation.JointsGroupNam
e? = nil) ->
[HumanBodyPose3DObservation.JointName :
```

```
Joint3D]

    /// Returns a 2D point for the joint
name you specify, relative to the input
image.
    ///
    /// - Parameters:
    ///   - jointName: The name of the
human body joint.
    /// - Returns: A projection of the 3D
position onto the original 2D image in
normalized, lower left origin
coordinates.
    public func pointInImage(for
jointName:
HumanBodyPose3DObservation.JointName) ->
NormalizedPoint

    /// Returns the parent joint of the
joint name you specify.
    public func parentJointName(for
jointName:
HumanBodyPose3DObservation.JointName) ->
HumanBodyPose3DObservation.JointName

    /// Returns a position relative to
the camera for the body joint you
specify.
    ///
    /// - Parameters:
    ///   - jointName: The name of the
human body joint.
    /// - Returns: The joint position, in
```

```
meters.
    public func
cameraRelativePosition(for jointName:
HumanBodyPose3DObservation.JointName) ->
simd_float4x4

    /// The supported joint names for the
body pose.
    public enum JointName : String,
Hashable, Sendable, Codable {

        /// A joint name that represents
the top of the head.
        case topHead

        /// A joint name that represents
the center of the head.
        case centerHead

        /// A joint name that represents
the point between the shoulders.
        case centerShoulder

        /// A joint name that represents
the left shoulder.
        case leftShoulder

        /// A joint name that represents
the right shoulder.
        case rightShoulder

        /// A joint name that represents
the left elbow.
```

```
        case leftElbow

        /// A joint name that represents
the right elbow.
        case rightElbow

        /// A joint name that represents
the left wrist.
        case leftWrist

        /// A joint name that represents
the right wrist.
        case rightWrist

        /// A joint name that represents
the left hip.
        case leftHip

        /// A joint name that represents
the right hip.
        case rightHip

        /// A joint name that represents
the left knee.
        case leftKnee

        /// A joint name that represents
the right knee.
        case rightKnee

        /// A joint name that represents
the left ankle.
        case leftAnkle
        /// A joint name that represents
the right ankle.
        case rightAnkle

        /// A joint name that represents
the point between the left hip and right
hip.
        case root

        /// A joint name that represents
the spine.
        case spine

        /// 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: String)

        /// 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, *)
        public typealias RawValue =
String

        /// 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: String { get
}
    }

    /// The supported joint group names
for the body pose.
    public enum JointsGroupName : String,
CaseIterable, Hashable, Sendable {

        /// A group name that represents
the head joints.
        case head

        /// A group name that represents
the left arm joints.
        case leftArm

        /// A group name that represents
```

```
the left leg joints.
        case leftLeg

        /// A group name that represents
the right arm joints.
        case rightArm

        /// A group name that represents
the right leg joints.
        case rightLeg

        /// A group name that represents
the torso joints.
        case torso

        /// 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: String)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[HumanBodyPose3DObservation.JointsGroupNa
me]

        /// 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, *)
        public typealias RawValue =
String
```

```
        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[HumanBodyPose3DObservation.JointsGroupNa
me] { get }

        /// 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: 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:
HumanBodyPose3DObservation, b:
HumanBodyPose3DObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
```

```
    public var hashValue: Int { get }
}

extension HumanBodyPose3DObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNHumanBodyPose3DObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension HumanBodyPose3DObservation :
Codable {

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
```

```
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
HumanBodyPose3DObservation.JointName :
RawRepresentable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
HumanBodyPose3DObservation.JointsGroupNam
e : RawRepresentable {
}

/// An observation that provides the body
points the analysis recognized.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct HumanBodyPoseObservation :
VisionObservation, PoseProviding {
    public var keypoints: MLMultiArray {
get throws }

    /// Retrieves a dictionary of joints
for a joint group.
    public func allJoints(in groupName:
HumanBodyPoseObservation.PoseJointsGroupN
ame? = nil) ->
[HumanBodyPoseObservation.PoseJointName :
Joint]

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
```

```
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
```

```
    public var description: String {
get }

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    public typealias PoseJointName =
HumanBodyPoseObservation.JointName

    public typealias PoseJointsGroupName
=
HumanBodyPoseObservation.JointsGroupName

    /// The supported joint names for the
body pose.
    public enum JointName : String,
Hashable, Sendable, Codable {

        /// The left ear.
        case leftEar

        /// The left eye.
        case leftEye

        /// The right ear.
        case rightEar

        /// The right eye.
        case rightEye
```

```swift
/// The neck.
case neck

/// The nose.
case nose

/// The left shoulder.
case leftShoulder

/// The left elbow.
case leftElbow

/// The left wrist.
case leftWrist

/// The right shoulder.
case rightShoulder

/// The right elbow.
case rightElbow

/// The right wrist.
case rightWrist

/// The root (waist).
case root

/// The left hip.
case leftHip

///  The left knee.
case leftKnee
```

```
        /// The left ankle.
        case leftAnkle

        /// The right hip.
        case rightHip

        /// The right knee.
        case rightKnee

        /// The right ankle.
        case rightAnkle

        /// 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: String)

        /// 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, *)
        public typealias RawValue =
String

        /// 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: String { get
}
    }

    /// The supported joint group names
for the body pose.
    public enum JointsGroupName : String,
CaseIterable, Hashable, Sendable {

        /// The face.
        case face

        /// The torso.
        case torso

        /// The left arm.
        case leftArm

        /// The right arm.
        case rightArm
        /// The left leg.
        case leftLeg

        /// The right leg.
        case rightLeg

        /// 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: String)
        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[HumanBodyPoseObservation.JointsGroupName
]

        /// 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, *)
        public typealias RawValue =
String

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[HumanBodyPoseObservation.JointsGroupName
] { get }
```

```
        /// 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: String { get
}
    }

    public let leftHand:
HumanHandPoseObservation?

    public let rightHand:
HumanHandPoseObservation?
```

```
    /// 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:
HumanBodyPoseObservation, b:
HumanBodyPoseObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}
```

```swift
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension HumanBodyPoseObservation :
Codable {

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws
}

extension HumanBodyPoseObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNHumanBodyPoseObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
HumanBodyPoseObservation.JointName :
RawRepresentable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
HumanBodyPoseObservation.JointsGroupName
: RawRepresentable {
}

/// An observation that provides the hand
points the analysis recognized.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct HumanHandPoseObservation :
VisionObservation, PoseProviding {

    public enum Chirality : Codable,
Equatable, Hashable, Sendable {

        case left

        case right

        /// 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:
HumanHandPoseObservation.Chirality, b:
HumanHandPoseObservation.Chirality) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
```

```
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
```

```
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
```

```
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// The chirality, or handedness, of
a pose.
    public let chirality:
HumanHandPoseObservation.Chirality?

    /// Retrieves a dictionary of joints
for a joint group.
    public func allJoints(in groupName:
HumanHandPoseObservation.PoseJointsGroupN
ame? = nil) ->
[HumanHandPoseObservation.PoseJointName :
Joint]

    public var keypoints: MLMultiArray {
get throws }

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
```

```
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    public typealias PoseJointName =
HumanHandPoseObservation.JointName

    public typealias PoseJointsGroupName
=
HumanHandPoseObservation.JointsGroupName

    /// The supported joint names for the
hand pose.
    public enum JointName : String,
Hashable, Sendable, Codable {

        /// The tip of the thumb.
        case thumbTip
        /// The thumb’s interphalangeal
(IP) joint.
        case thumbIP

        /// The thumb’s
metacarpophalangeal (MP) joint.
        case thumbMP

        /// The thumb’s carpometacarpal
(CMC) joint.
        case thumbCMC

        ///  The tip of the index finger.
        case indexTip

        ///  The index finger’s distal
interphalangeal (DIP) joint.
        case indexDIP

        /// The index finger’s proximal
interphalangeal (PIP) joint.
        case indexPIP

        /// The index finger’s
metacarpophalangeal (MCP) joint.
        case indexMCP

        /// The tip of the middle finger.
        case middleTip

        /// The middle finger’s distal
interphalangeal (DIP) joint.
        case middleDIP
        /// The middle finger’s proximal
interphalangeal (PIP) joint.
        case middlePIP

        /// The middle finger’s
metacarpophalangeal (MCP) joint.
        case middleMCP

        /// The tip of the ring finger.
        case ringTip

        /// The ring finger’s distal
interphalangeal (DIP) joint.
        case ringDIP

        /// The ring finger’s proximal
interphalangeal (PIP) joint.
        case ringPIP

        /// The ring finger’s
metacarpophalangeal (MCP) joint.
        case ringMCP

        /// The tip of the little finger.
        case littleTip

        /// The little finger’s distal
interphalangeal (DIP) joint.
        case littleDIP

        /// The little finger’s proximal
interphalangeal (PIP) joint.
```

```
        case littlePIP

        /// The little finger’s
metacarpophalangeal (MCP) joint.
        case littleMCP

        /// The wrist.
        case wrist

        /// 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: String)

        /// 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, *)
        public typealias RawValue =
String

        /// 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: String { get
}
    }

    /// The joint group names available
in the observation.
    public enum JointsGroupName : String,
CaseIterable, Hashable, Sendable {

        /// The thumb.
        case thumb

        /// The index finger.
        case indexFinger

        /// The little finger.
        case littleFinger

        /// The middle finger.
        case middleFinger

        /// The ring finger.
        case ringFinger
        /// 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: String)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[HumanHandPoseObservation.JointsGroupName
]

        /// 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, *)
        public typealias RawValue =
String

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[HumanHandPoseObservation.JointsGroupName
] { get }

        /// 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: 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:
HumanHandPoseObservation, b:
HumanHandPoseObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension HumanHandPoseObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension HumanHandPoseObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNHumanHandPoseObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
HumanHandPoseObservation.JointName :
RawRepresentable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
HumanHandPoseObservation.JointsGroupName
: RawRepresentable {
}

/// An object that represents a person
that the request detects.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct HumanObservation :
VisionObservation, BoundingBoxProviding {

    /// A Boolean value that indicates
whether the observation represents an
upper-body or full-body rectangle.
    public let isUpperBodyOnly: Bool

    /// The bounding box of the object.
    ///
    /// The coordinate system is
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    public let boundingBox:
NormalizedRect
```

```
    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
HumanObservation, b: HumanObservation) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension HumanObservation : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}
```

```
/// HumanObservstion Initializers
extension HumanObservation {

    public init(boundingBox:
NormalizedRect, revision:
DetectHumanRectanglesRequest.Revision? =
nil)
}

extension HumanObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNHumanObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
ImageAestheticsScoresObservation :
VisionObservation {

    /// `isUtility` represents images
that are not necessarily of poor image
quality but may not have memorable or
exciting content.
    public let isUtility: Bool

    /// A score which incorporates
aesthetic score, failure score and
utility labels.
    ///
    /// `overallScore` is within the
range [−1, 1] where 1 is most desirable
and −1 is not desirable.
    public let overallScore: Float

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
```

```
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
```

```
RequestDescriptor?

    /// 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:
ImageAestheticsScoresObservation, b:
ImageAestheticsScoresObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
```

```
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
ImageAestheticsScoresObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension
ImageAestheticsScoresObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNImageAestheticsScoresObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
ImageCoordinateConversionHelpers {

    public static func
imagePointForNormalizedPoint(normalizedPo
int: CGPoint, imageSize: CGSize) ->
CGPoint

    public static func
verticallyFlippedImagePoint(imagePoint:
CGPoint, imageHeight: UInt32) -> CGPoint

    public static func
verticallyFlippedNormalizedPoint(normaliz
edPoint: CGPoint) -> CGPoint
```

```
    public static func
imageRectForNormalizedRect(normalizedRect
: CGRect, imageSize: CGSize) -> CGRect

    public static func
verticallyFlippedImageRect(imageRect:
CGRect, imageHeight: UInt32) -> CGRect

    public static func
verticallyFlippedNormalizedRect(normalize
dRect: CGRect, imageHeight: UInt32) ->
CGRect
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum ImageCropAndScaleAction :
CaseIterable, Codable, Equatable,
Hashable, Sendable {

    /// Scale image maintaining aspect
ratio to fit on the short side and crop
centered on the long side.
    case centerCrop

    /// Scale to size required by
algorithm while maintaining the original
aspect ratio.
    case scaleToFit

    /// An option that scales the image
to fill the input dimensions, resizing it
if necessary.
```

```
    case scaleToFill

    /// Scale image maintaining aspect
ratio to fit on the long side but also
rotate by 90 degrees counter clockwise to
optimize portrait images to fit into
landscape buffers for algorithms that are
rotation agnostic.
    case scaleToFitPlus90CCWRotation

    /// Scale image and rotate by 90
degrees counter clockwise to optimize
portrait images to fill into landscape
buffers for algorithms that are rotation
agnostic.
    case scaleToFillPlus90CCWRotation

    /// 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:
ImageCropAndScaleAction, b:
ImageCropAndScaleAction) -> Bool
```

```
    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// A type that can represent a
collection of all values of this type.
    @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
    public typealias AllCases =
[ImageCropAndScaleAction]
```

```
    /// A collection of all values of
this type.
    nonisolated public static var
allCases: [ImageCropAndScaleAction] { get
}

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
```

```
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/// An object that represents a
perspective warp transformation.
///
/// This type of observation results from
a
``TrackHomographicImageRegistrationReques
t``, informing the `warpTransform`
performed to align the input images.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
ImageHomographicAlignmentObservation :
```

```
VisionObservation {

    /// The warp transform matrix to
morph the floating image into the
reference image.
    public let warpTransform:
matrix_float3x3

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation's start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
```

```
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
```

```
    /// `Point` type's `description`
property.
    public var description: String {
get }

    public func applyTransform(to
ciImage: CIImage) -> CIImage

    /// 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:
ImageHomographicAlignmentObservation, b:
ImageHomographicAlignmentObservation) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
```

```
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
ImageHomographicAlignmentObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
```

```
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension
ImageHomographicAlignmentObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNImageHomographicAlignmentObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct ImagePixelDimensions {
}

/// An image analysis request that
operates on a region of interest and
produces observations.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol ImageProcessingRequest :
```

```
VisionRequest {

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    var regionOfInterest: NormalizedRect
{ get set }

    /// Perform the request on a
`CVPixelBuffer` and produce observations.
    ///
    /// - Parameters:
    ///   - pixelBuffer: The input
`CVPixelBuffer` on which to perform the
request.
    ///   - orientation: The orientation
of the input image. Default is `nil`.
    ///
    /// - Returns: The observation(s)
produced by the request.
    func perform(on pixelBuffer:
CVPixelBuffer, orientation:
CGImagePropertyOrientation?) async throws
-> Self.Result

    /// Perform the request on an image
`URL` and produce observations.
```

```
    ///
    /// - Parameters:
    ///   - url: The input `URL` on which
to perform the request.
    ///   - orientation: The orientation
of the input image. Default is `nil`.
    ///
    /// - Returns: The observation(s)
produced by the request.
    func perform(on url: URL,
orientation: CGImagePropertyOrientation?)
async throws -> Self.Result

    /// Perform the request on a
`CGImage` and produce observations.
    ///
    /// - Parameters:
    ///   - image: The input `CGImage` on
which to perform the request.
    ///   - orientation: The orientation
of the input image. Default is `nil`.
    ///
    /// - Returns: The observation(s)
produced by the request.
    func perform(on image: CGImage,
orientation: CGImagePropertyOrientation?)
async throws -> Self.Result

    /// Perform the request on a
`CIImage` and produce observations.
    ///
    /// - Parameters:
    ///   - image: The input `CIImage` on
```

```
which to perform the request.
    ///   - orientation: The orientation
of the input image. Default is `nil`.
    ///
    /// - Returns: The observation(s)
produced by the request.
    func perform(on image: CIImage,
orientation: CGImagePropertyOrientation?)
async throws -> Self.Result

    /// Perform the request on a
`CMSampleBuffer` and produce
observations.
    ///
    /// - Parameters:
    ///   - sampleBuffer: The input
`CMSampleBuffer` on which to perform the
request.
    ///   - orientation: The orientation
of the input image. Default is `nil`.
    ///
    /// - Returns: The observation(s)
produced by the request.
    func perform(on sampleBuffer:
CMSampleBuffer, orientation:
CGImagePropertyOrientation?) async throws
-> Self.Result

    /// Perform the request on image
`Data` and produce observations.
    ///
    /// - Parameters:
    ///   - data: The input `Data` on
```

```
which to perform the request.
    ///   - orientation: The orientation
of the input image. Default is `nil`.
    ///
    /// - Returns: The observation(s)
produced by the request.
    func perform(on data: Data,
orientation: CGImagePropertyOrientation?)
async throws -> Self.Result
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension ImageProcessingRequest {

    public func perform(on pixelBuffer:
CVPixelBuffer, orientation:
CGImagePropertyOrientation? = nil) async
throws -> Self.Result

    public func perform(on url: URL,
orientation: CGImagePropertyOrientation?
= nil) async throws -> Self.Result

    public func perform(on image:
CGImage, orientation:
CGImagePropertyOrientation? = nil) async
throws -> Self.Result

    public func perform(on image:
CIImage, orientation:
CGImagePropertyOrientation? = nil) async
throws -> Self.Result
```

```
    public func perform(on sampleBuffer:
CMSampleBuffer, orientation:
CGImagePropertyOrientation? = nil) async
throws -> Self.Result

    public func perform(on data: Data,
orientation: CGImagePropertyOrientation?
= nil) async throws -> Self.Result
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class ImageRequestHandler :
@unchecked Sendable {

    public convenience init(_ imageURL:
URL, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(_ image:
CGImage, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(_ image:
CIImage, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(_
pixelBuffer: CVPixelBuffer, depthData:
AVDepthData? = nil, orientation:
CGImagePropertyOrientation? = nil)
```

```
    public convenience init(_
sampleBuffer: CMSampleBuffer, depthData:
AVDepthData? = nil, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(_ data: Data,
orientation: CGImagePropertyOrientation?
= nil)

    /// Perform a Vision request on the
handler's image.
    final public func perform<T>(_
request: T) async throws -> T.Result
where T : VisionRequest

    /// Perform one or more Vision
requests on the handler's image.
    ///
    /// The function returns only after
all requests have completed.
    final public func perform<each T>(_
request: repeat each T) async throws ->
(repeat (each T).Result) where repeat
each T : VisionRequest

    final public func performAll(_
requests: some Collection<any
VisionRequest>) -> some
AsyncSequence<VisionResult, Never>

}

/// Affine transform information that an
```

```
image alignment request produces.
///
/// This type of observation results from
a
``TrackTranslationalImageRegistrationRequ
est``, informing the `alignmentTransform`
performed to align the input images.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct
ImageTranslationAlignmentObservation :
VisionObservation {

    /// The alignment transform to align
the floating image with the reference
image.
    public let alignmentTransform:
CGAffineTransform

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float
    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
```

```
    ///                 return "(\(x), \(y))"
    ///             }
    ///         }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    public func applyTransform(to
ciImage: CIImage) -> CIImage

    public init(_ observation:
VNImageTranslationAlignmentObservation)

    /// 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:
ImageTranslationAlignmentObservation, b:
ImageTranslationAlignmentObservation) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
ImageTranslationAlignmentObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/// An observation that contains an
instance mask that labels instances in
the mask.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct InstanceMaskObservation :
```

```
VisionObservation, @unchecked Sendable {

    /// The collection that contains all
instances, excluding the background.
    public let allInstances: IndexSet

    /// The resulting mask that
represents all instances.
    ///
    /// A pixel can only correspond to
one instance. A 0 represents the
background, and all other values
represent the indices of the instances.
    public let allInstancesMask:
PixelBufferObservation

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Creates a low-resolution mask
from the instances you specify.
    ///
    /// - Parameters:
    ///   - instances: An IndexSet of
selected instances, where 0 is the
background.
    ///
    /// - Returns: The pixel buffer that
contains the mask.
    public func generateMask(for
instances: IndexSet) throws ->
CVPixelBuffer

    /// Creates a high-resolution image
with everything except for the instances
you specify masked out.
    ///
    /// The image produced has the same
resolution as the image in the
`requestHandler`.
    ///
    /// - Parameters:
    ///   - instances: An IndexSet of
selected instances, where 0 is the
background.
```

```
    ///   - requestHandler: A request
handler containing an image to be masked.
    ///   - croppedToInstancesExtent:
Crops the image to the smallest rectangle
containg all instances
    /// - Returns: The pixel buffer that
contains the image.
    public func generateMaskedImage(for
instances: IndexSet, imageFrom
requestHandler: ImageRequestHandler,
croppedToInstancesExtent: Bool = false)
throws -> CVPixelBuffer

    /// Creates a high-resolution mask
representing a combination of the
instances you specify.
    ///
    /// The image produced has the same
resolution as the image in the
`requestHandler`.
    ///
    /// - Parameters:
    ///   - instances: An IndexSet of
selected instances, where 0 is the
background.
    ///   - requestHandler: A request
handler containing an image.
    ///
    /// - Returns: The pixel buffer that
contains the mask.
    public func generateScaledMask(for
instances: IndexSet, scaledToImageFrom
requestHandler: ImageRequestHandler)
```

```
throws -> CVPixelBuffer

    /// Creates an index set containing
the instance found at the specified
point.
    public func instanceAtPoint(_ point:
NormalizedPoint) -> IndexSet

    /// 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 == (lhs:
InstanceMaskObservation, rhs:
InstanceMaskObservation) -> Bool

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension InstanceMaskObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
```

```
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension InstanceMaskObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init?(_ observation:
VNInstanceMaskObservation)
}

/// A body pose joint represented as a
normalized point in an image, along with
a joint name label and a confidence
value.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct Joint : Codable, Equatable,
Hashable, Sendable,
CustomStringConvertible {

    /// The location of the joint in
normalized coordinates.
    public let location: NormalizedPoint

    /// The joint's identifier label.
    public let jointName: String

    /// A confidence score that indicates
the detected joint’s accuracy.
    public let confidence: Float

    public func distance(to joint: Joint)
-> CGFloat

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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: Joint, b:
Joint) -> Bool

    /// Hashes the essential components
```

```
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
```

```
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct Joint3D : Codable,
Equatable, Hashable, Sendable {

    public let position: simd_float4x4

    public let localPosition:
simd_float4x4

    public let identifier: String

    public let parentJoint: String

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
```

```
    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    public init(position: simd_float4x4,
localPosition: simd_float4x4, identifer:
String, parentJoint: String)

    /// 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: Joint3D, b:
Joint3D) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

public typealias
NamedMultipleObjectDataAccessBlock = (_
namedObjectDataMap: [String : Data])
throws -> Void

public typealias
NamedObjectDataAccessBlock = (_
objectData: Data) throws -> Void

public typealias NamedObjectsDictionary =
[String : Any]

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct NormalizedCircle {

    public init(center: NormalizedPoint,
radius: CGFloat)
```

```
    public let center: NormalizedPoint

    public let radius: CGFloat

    /// Determines if this circle,
including its boundary, contains the
specified point.
    public func contains(_ point:
NormalizedPoint) -> Bool

    /// Determines if a ring around this
circle's circumference contains the
specified point.
    public func contains(_ point:
NormalizedPoint,
inCircumferentialRingOfWidth ringWidth:
CGFloat) -> Bool

    public static var zero:
NormalizedCircle { get }

    public static func boundingCircle(for
points: [NormalizedPoint]) ->
NormalizedCircle
}

/// A 2D point with x and y coordinates
in the range [0, 1].
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct NormalizedPoint : Hashable,
Equatable, Codable, Sendable,
CustomStringConvertible {

    /// Creates a point object with the
specified coordinates.
    ///
    /// - Parameters:
    ///   - x: The x-coordinate value.
    ///   - y: The y-coordinate value.
    public init(x: CGFloat, y: CGFloat)

    /// Creates a point object from the
specified Core Graphics point.
    public init(normalizedPoint: CGPoint)

    /// Creates a normalized point from a
point in an image coordinate space.
    ///
    /// - Parameters:
    ///   - imagePoint: A point in the
image coordinate space.
    ///   - imageSize: The size of the
image.
    public init(imagePoint: CGPoint, in
imageSize: CGSize)

    /// Creates a point normalized to a
region of interest within an image.
    ///
    /// - Parameters:
    ///   - imagePoint: A point in the
image coordinate space.
    ///   - imageSize: The size of the
image.
```

```
    ///   - regionOfInterest: The region
within the image that the point will be
normalized to
    public init(imagePoint: CGPoint, in
imageSize: CGSize, normalizedTo
regionOfInterest: NormalizedRect)

    /// A point object that represents
the origin, [0,0].
    public static var zero:
NormalizedPoint { get }

    /// The Core Graphics point for this
point.
    public let cgPoint: CGPoint

    /// The x-coordinate.
    public var x: CGFloat { get }

    /// The y-coordinate.
    public var y: CGFloat { get }

    /// Converts a point normalized to a
region within an image into full image
coordinates.
    ///
    /// - Parameters:
    ///   - imageSize: The size of the
image.
    ///   - regionOfInterest: The region
within an image the `NormalizedPoint` is
normalized to.
    public func toImageCoordinates(from
regionOfInterest: NormalizedRect,
imageSize: CGSize, origin:
CoordinateOrigin = .lowerLeft) -> CGPoint

    /// Converts a point in normalized
coordinates into image coordinates.
    ///
    /// - Parameters:
    ///   - imageSize: The size of the
image.
    public func toImageCoordinates(_
imageSize: CGSize, origin:
CoordinateOrigin = .lowerLeft) -> CGPoint

    /// Returns a `NormalizedPoint` with
the origin flipped between the top and
bottom of the image.
    public func verticallyFlipped() ->
NormalizedPoint

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
```

```
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
NormalizedPoint, b: NormalizedPoint) ->
Bool

    /// Encodes this value into the given
encoder.
```

```
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
```

```
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/// A rectangle with normalized
coordinates in the range [0, 1].
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct NormalizedRect : Equatable,
Hashable, Codable, Sendable,
CustomStringConvertible {

    /// Creates a rectangle with the
specified coordinates.
    ///
    /// - Parameters:
    ///   - x: The x-coordinate of the
rectangle's lower-left corner.
    ///   - y: The y-coordinate of the
rectangle's lower-left corner..
    ///   - width: The width of the
rectangle
    ///   - height: The height of the
rectangle
    public init(x: CGFloat, y: CGFloat,
width: CGFloat, height: CGFloat)

    /// Creates a normalized rectangle
from a rectangle in an image coordinate
space.
    ///
    /// - Parameters:
    ///   - imageRect: A rectangle in the
image coordinate space.
    ///   - imageSize: The size of the
image.
    public init(imageRect: CGRect, in
imageSize: CGSize)

    /// Creates a rectangle normalized to
a region of interest in an image from a
rectangle in an image coordinate space.
    ///
    /// - Parameters:
    ///   - imageRect: A rectangle in the
image coordinate space.
    ///   - imageSize: The size of the
image.
    public init(imageRect: CGRect, in
imageSize: CGSize, normalizedTo
regionOfInterest: NormalizedRect)

    /// Creates a rectangle from the
specified Core Graphics rectangle.
    public init(normalizedRect: CGRect)

    /// A normalized rectangle with
origin at [0,0] and a width and height of
```

```
1.0.
    public static var fullImage:
NormalizedRect { get }

    /// The normalized rectangle as a
CGRect.
    public let cgRect: CGRect

    /// The lower-left hand corner of the
rectangle.
    public var origin: CGPoint { get }

    /// The width of the rectangle
    public var width: CGFloat { get }

    /// The height of the rectangle
    public var height: CGFloat { get }

    /// Converts a rectangle in
normalized coordinates into image
coordinates.
    ///
    /// - Parameters:
    ///   - imageSize: The size of the
image.
    public func toImageCoordinates(_
imageSize: CGSize, origin:
CoordinateOrigin = .lowerLeft) -> CGRect

    /// Converts a rectangle normalized
to a region within an image into full
image coordinates.
    ///
    /// - Parameters:
    ///   - imageSize: The size of the
image.
    ///   - regionOfInterest: The region
within an image the `NormalizedPoint` is
normalized to.
    public func toImageCoordinates(from
regionOfInterest: NormalizedRect,
imageSize: CGSize, origin:
CoordinateOrigin = .lowerLeft) -> CGRect

    /// Returns a `NormalizedRect` with
the origin flipped between the top and
bottom of the image.
    public func verticallyFlipped() ->
NormalizedRect

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
NormalizedRect, b: NormalizedRect) ->
Bool

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
```

```
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
```

```
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

/// An object that represents an optical
flow that an image analysis request
produces.
///
/// The optical flow is a 2d image, with
each pixel representing the directional
change from a previous to current image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct OpticalFlowObservation :
VisionObservation, @unchecked Sendable {

    /// The size of the observation
image.
    public var size: CGSize { get }

    /// The four-character code OSType
identifier for the pixel format.
    public var pixelFormat: OSType {
get }

    /// The unique identifier for the
observation.
    public let uuid: UUID
    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
```

```
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    public func withUnsafePointer<R>(_
body: (UnsafeRawPointer) -> R) -> R

    /// Returns the optical flow for the
specified location in the observation
image
```

```
    public func flow(at point:
NormalizedPoint) -> (Float, Float)

    /// 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 == (lhs:
OpticalFlowObservation, rhs:
OpticalFlowObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
```

```
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

extension OpticalFlowObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init?(_ observation:
VNPixelBufferObservation)
}

/// An object that represents an image
that an image analysis request produces.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct PixelBufferObservation :
VisionObservation, @unchecked Sendable {

    /// The size of the observation
image.
    public var size: CGSize { get }

    /// The four-character code OSType
identifier for the pixel format.
    public var pixelFormat: OSType {
get }

    /// A Core Graphics image created
from the pixel buffer observation.
    public var cgImage: CGImage { get
throws }
```

```
    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
```

```
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    public func withUnsafePointer<R>(_
body: (UnsafeRawPointer) -> R) -> R

    /// Returns the pixel data for the
specified location in the image
    public func pixel(at point:
NormalizedPoint) -> Float

    /// 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 == (lhs:
PixelBufferObservation, rhs:
PixelBufferObservation) -> Bool

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
```

```
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

extension PixelBufferObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init?(_ observation:
VNPixelBufferObservation)
}

/// An observation that provides a
collection of joints that make up a pose.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol PoseProviding {

    associatedtype PoseJointName :
Decodable, Encodable, Hashable,
RawRepresentable where
Self.PoseJointName.RawValue == String

    associatedtype PoseJointsGroupName :
CaseIterable, RawRepresentable where
Self.PoseJointsGroupName.RawValue ==
String

    /// The names of the available joints
in the observation.
    var availableJointNames:
```

```
[Self.PoseJointName] { get }

    /// The names of the available joint
groupings in the observation.
    var availableJointsGroupNames:
[Self.PoseJointsGroupName] { get }

    /// Retrieves a  joint for a given
joint name.
    func joint(for jointName:
Self.PoseJointName) -> Joint?

    /// Retrieves a dictionary of joints
for a joint group.
    func allJoints(in groupName:
Self.PoseJointsGroupName?) ->
[Self.PoseJointName : Joint]
}

/// An protocol for objects that have a
bounding quadrilateral.
///
/// The quadrilateral's coordinate system
is normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol QuadrilateralProviding :
BoundingBoxProviding {

    /// The coordinates of the upper-left
corner of the quadrilateral.
```

```
    var topLeft: NormalizedPoint { get }

    /// The coordinates of the upper-
right corner of the quadrilateral.
    var topRight: NormalizedPoint { get }

    /// The coordinates of the lower-
right corner of the quadrilateral.
    var bottomRight: NormalizedPoint {
get }

    /// The coordinates of the lower-left
corner of the quadrilateral.
    var bottomLeft: NormalizedPoint { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension QuadrilateralProviding {

    /// The bounding box of the object.
    ///
    /// The coordinate system is
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    public var boundingBox:
NormalizedRect { get }
}

/// A request that will recognize various
animals in an image.
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct RecognizeAnimalsRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[RecognizedObjectObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
```

```
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
RecognizeAnimalsRequest.Revision, b:
RecognizeAnimalsRequest.Revision) -> 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:
RecognizeAnimalsRequest.Revision, b:
RecognizeAnimalsRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
```

```
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
```

```
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
```

```
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
RecognizeAnimalsRequest.Revision? = nil)

    public enum Animal : String,
Sendable, Equatable, Codable, Hashable {

        case dog

        case cat

        /// 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: String)

        /// 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, *)
        public typealias RawValue =
String

        /// 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: String { get
}
    }

    /// The collection of animal
identifiers that the request can detect.
    public var supportedAnimals:
[RecognizeAnimalsRequest.Animal] { get }

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
```

```
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
RecognizeAnimalsRequest.Revision

    /// The collection of currently-
supported revisions for
`RecognizeAnimalsRequest`.
    public static let supportedRevisions:
[RecognizeAnimalsRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
```

```
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
RecognizeAnimalsRequest, b:
RecognizeAnimalsRequest) -> Bool
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension
RecognizeAnimalsRequest.Animal :
RawRepresentable {
}

/// An image analysis request that finds
projected rectangular regions in an
image.
///
/// A rectangle detection request locates
regions of an image with rectangular
shape, like credit cards, business cards,
documents, and signs. The request returns
its observations in the form of
``RectangleObservation`` objects, which
contain normalized coordinates of
bounding boxes containing the rectangle.
/// Use this type of request to find the
bounding boxes of rectangles in an image.
Vision returns observations for
rectangles found in all orientations and
sizes, along with a confidence level to
indicate how likely it's that the
observation contains an actual rectangle.
/// To further configure or restrict the
types of rectangles found, set properties
on the request specifying a range of
aspect ratios, sizes, and quadrature
tolerance.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct RecognizeTextRequest :
ImageProcessingRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
[RecognizedTextObservation]

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {
        case revision3

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
RecognizeTextRequest.Revision, b:
RecognizeTextRequest.Revision) -> 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:
RecognizeTextRequest.Revision, b:
RecognizeTextRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
```

```
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
```

```
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - revision: The specific
algorithm or implementation revision that
is to be used to perform the request.
    public init(_ revision:
RecognizeTextRequest.Revision? = nil)

    /// Constants that identify the
performance and accuracy of the text
recognition.
    public enum RecognitionLevel :
```

```
CaseIterable, Sendable, Equatable,
Codable, Hashable {

        /// Accurate text recognition
takes more time to produce a more
comprehensive result.
        case accurate

        /// Fast text recognition returns
results more quickly at the expense of
accuracy.
        case fast

        /// 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:
RecognizeTextRequest.RecognitionLevel, b:
RecognizeTextRequest.RecognitionLevel) ->
Bool

        /// Hashes the essential
components of this value by feeding them
into the
```

```
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[RecognizeTextRequest.RecognitionLevel]

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[RecognizeTextRequest.RecognitionLevel] {
get }

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
```

```
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// The minimum height, relative to
the image height, of the text to
recognize.
    ///
    /// Specify a floating-point number
relative to the image height. For
example, to limit recognition to text
```

```
that’s half of the image height, use 0.5.
Increasing the size reduces memory
consumption and expedites recognition
with the tradeoff of ignoring text
smaller than the minimum height. The
default value is 1/32, or 0.03125.
    public var minimumTextHeightFraction:
Float

    /// A value that determines whether
the request prioritizes accuracy or speed
in text recognition.
    public var recognitionLevel:
RecognizeTextRequest.RecognitionLevel

    ///  A Boolean value that indicates
whether to attempt detecting the language
to use the appropriate model for
recognition and language correction.
    public var
automaticallyDetectsLanguage: Bool

    /// An array of languages to detect,
in priority order.
    ///
    /// The order of the languages in the
array defines the order in which
languages are used during language
processing and text recognition.
    public var recognitionLanguages:
[Locale.Language]

    ///  A Boolean value that indicates
```

```
whether the request applies language
correction during the recognition
process.
    ///
    ///  When this value is `true`,
Vision applies language correction during
the recognition process. Disabling this
property returns the raw recognition
results, which provides performance
benefits but less accurate results.
    public var usesLanguageCorrection:
Bool

    /// An array of strings to supplement
the recognized languages at the word-
recognition stage.
    ///
    /// Custom words take precedence over
the standard lexicon. The request ignores
this value if `usesLanguageCorrection` is
`false`.
    public var customWords: [String]

    /// The identifiers of the languages
that the request supports.
    public var
supportedRecognitionLanguages:
[Locale.Language] { get }

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
```

```
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    public let revision:
RecognizeTextRequest.Revision

    ///  The collection of currently-
supported revisions for
`RecognizeTextRequest`.
    public static let supportedRevisions:
[RecognizeTextRequest.Revision]

    /// An enum that identifies the
request and request revision.
    public var descriptor:
RequestDescriptor { get }

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
```

```
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// 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:
RecognizeTextRequest, b:
RecognizeTextRequest) -> Bool
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An observation with an array of
classification labels that classify the
recognized object.
///
/// The confidence of the classifications
sum up to 1.0. Multiply the
classification confidence with the
confidence of this observation.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct RecognizedObjectObservation
: VisionObservation, BoundingBoxProviding
{
    /// The classification(s) of the
recognized object.
    public let labels:
[ClassificationObservation]

    /// The bounding box of the object.
    ///
    /// The coordinate system is
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    public let boundingBox:
NormalizedRect

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
```

```
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
RecognizedObjectObservation, b:
RecognizedObjectObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
```

```
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension RecognizedObjectObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws
```

```
    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension RecognizedObjectObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNRecognizedObjectObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct RecognizedText : Equatable,
Hashable, @unchecked Sendable,
CustomStringConvertible {

    /// The top candidate for recognized
text.
    public var string: String { get }

    /// A normalized confidence score for
```

```
the text recognition result.
    public var confidence: Float { get }

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
```

```
property.
    public var description: String {
get }

    /// Calculates the bounding box
around the characters in the range of a
string.
    ///
    /// Bounding boxes aren’t always an
exact fit around the characters. Use them
to display in user interfaces to provide
general guidance,
    /// but avoid using their contents
for image processing.
    ///
    /// - Parameters:
    ///   - range: The range of the
characters in the text string to draw a
bounding box around.
    public func boundingBox(for range:
Range<String.Index>) ->
RectangleObservation?

    /// 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:
RecognizedText, b: RecognizedText) ->
Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)
```

```
    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension RecognizedText : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct RecognizedTextObservation :
VisionObservation, QuadrilateralProviding
{

    /// The coordinates of the upper-left
corner of the quadrilateral.
    public var topLeft: NormalizedPoint

    /// The coordinates of the upper-
right corner of the quadrilateral.
    public var topRight: NormalizedPoint
    /// The coordinates of the lower-
right corner of the quadrilateral.
    public var bottomRight:
NormalizedPoint

    /// The coordinates of the lower-left
corner of the quadrilateral.
    public var bottomLeft:
NormalizedPoint

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?
    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// Requests the n top candidates for
a recognized text string.
    ///
    /// This function returns no more
than n candidates, but it may return
fewer than n candidates.
    ///
    /// - Parameters:
    ///     - maxCandidateCount: The
maximum number of candidates to return.
This can't exceed 10.
    /// - Returns: An array of the n top
candidates, sorted by decreasing
confidence score.
    public func topCandidates(_
maxCandidateCount: Int) ->
[RecognizedText]

    /// 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:
RecognizedTextObservation, b:
RecognizedTextObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension RecognizedTextObservation :
Codable {

    /// Encodes this value into the given
encoder.
```

```
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension RecognizedTextObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
```

```
VNRecognizedTextObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct RectangleObservation :
VisionObservation, QuadrilateralProviding
{

    public init(topLeft: NormalizedPoint,
topRight: NormalizedPoint, bottomRight:
NormalizedPoint, bottomLeft:
NormalizedPoint)

    /// The coordinates of the upper-left
corner of the quadrilateral.
    public let topLeft: NormalizedPoint

    /// The coordinates of the upper-
right corner of the quadrilateral.
    public let topRight: NormalizedPoint

    /// The coordinates of the lower-
right corner of the quadrilateral.
    public let bottomRight:
NormalizedPoint

    /// The coordinates of the lower-left
corner of the quadrilateral.
    public let bottomLeft:
NormalizedPoint

    /// The unique identifier for the
```

```
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
```

```
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
RectangleObservation, b:
RectangleObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension RectangleObservation : Codable
```

```
{

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension RectangleObservation {
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNRectangleObservation)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum RequestDescriptor :
CustomStringConvertible, Equatable,
Sendable, Codable, Hashable {

    case
detectFaceRectanglesRequest(DetectFaceRec
tanglesRequest.Revision)

    case
detectHumanRectanglesRequest(DetectHumanR
ectanglesRequest.Revision)

    case
classifyImageRequest(ClassifyImageRequest
.Revision)

    case
calculateImageAestheticsScoresRequest(Cal
culateImageAestheticsScoresRequest.Revisi
on)

    case
coreMLRequest(CoreMLRequest.Revision)
```

```
    case
detectAnimalBodyPoseRequest(DetectAnimalB
odyPoseRequest.Revision)

    case
detectBarcodesRequest(DetectBarcodesReque
st.Revision)

    case
detectContoursRequest(DetectContoursReque
st.Revision)

    case
detectDocumentSegmentationRequest(DetectD
ocumentSegmentationRequest.Revision)

    case
detectFaceCaptureQualityRequest(DetectFac
eCaptureQualityRequest.Revision)

    case
detectFaceLandmarksRequest(DetectFaceLand
marksRequest.Revision)

    case
detectHorizonRequest(DetectHorizonRequest
.Revision)

    case
detectHumanBodyPoseRequest(DetectHumanBod
yPoseRequest.Revision)

    case
```

```
detectHumanBodyPose3DRequest(DetectHumanB
odyPose3DRequest.Revision)

    case
detectHumanHandPoseRequest(DetectHumanHan
dPoseRequest.Revision)

    case
detectRectanglesRequest(DetectRectanglesR
equest.Revision)

    case
detectTextRectanglesRequest(DetectTextRec
tanglesRequest.Revision)

    case
detectTrajectoriesRequest(DetectTrajector
iesRequest.Revision)

    case
generateAttentionBasedSaliencyImageReques
t(GenerateAttentionBasedSaliencyImageRequ
est.Revision)

    case
generateImageFeaturePrintRequest(Generate
ImageFeaturePrintRequest.Revision)

    case
generateForegroundInstanceMaskRequest(Gen
erateForegroundInstanceMaskRequest.Revisi
on)
```

```
    case
generateObjectnessBasedSaliencyImageReque
st(GenerateObjectnessBasedSaliencyImageRe
quest.Revision)

    case
generatePersonSegmentationRequest(Generat
ePersonSegmentationRequest.Revision)

    case
generatePersonInstanceMaskRequest(Generat
ePersonInstanceMaskRequest.Revision)

    case
recognizeAnimalsRequest(RecognizeAnimalsR
equest.Revision)

    case
recognizeTextRequest(RecognizeTextRequest
.Revision)

    case
trackHomographicImageRegistrationRequest(
TrackHomographicImageRegistrationRequest.
Revision)

    case
trackObjectRequest(TrackObjectRequest.Rev
ision)

    case
trackOpticalFlowRequest(TrackOpticalFlowR
equest.Revision)
```

```
    case
trackRectangleRequest(TrackRectangleReque
st.Revision)

    case
trackTranslationalImageRegistrationReques
t(TrackTranslationalImageRegistrationRequ
est.Revision)

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
RequestDescriptor, b: RequestDescriptor)
-> Bool

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
```

```
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
```

```
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
```

```
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct ResourceVersion : Sendable,
Codable, Hashable, Equatable, Comparable,
CustomStringConvertible {

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws

    /// A textual representation of this
instance.
```

```
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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 == (lhs:
ResourceVersion, rhs: ResourceVersion) ->
Bool

    /// Returns a Boolean value
indicating whether the value of the first
    /// argument is less than that of the
second argument.
    ///
    /// This function is the only
requirement of the `Comparable` protocol.
The
    /// remainder of the relational
operator functions are implemented by the
    /// standard library for any type
that conforms to `Comparable`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func < (lhs:
```

```
ResourceVersion, rhs: ResourceVersion) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An observation that contains a
grayscale heat map of important areas
across an image.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct SaliencyImageObservation :
VisionObservation {

    /// A collection of objects
describing the distinct areas of the
saliency heat map.
    ///
    /// The objects in this array don't
follow any specific ordering.
    /// It's up to your app to iterate
across the observations and apply desired
ordering.
    public let salientObjects:
[RectangleObservation]

    /// A grayscale heat map of important
areas across the image.
    ///
    /// The heat map is a pixel buffer in
a one-component floating-point pixel
format.
    public let heatMap:
PixelBufferObservation

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
```

```
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// 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:
SaliencyImageObservation, b:
SaliencyImageObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension SaliencyImageObservation :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
```

```
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension SaliencyImageObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init?(_ observation:
VNSaliencyImageObservation)
}

/// A request type that builds evidence
of a condition over time.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol StatefulRequest :
VisionRequest {

    /// The minimum number of frames that
```

```
the request has to process on before
reporting back any observation.
    ///
    /// This information is provided by
the request once initialized with its
required paramters.
    /// Video-based requests often need a
minimum number of frames before they can
report back any observation.
    /// An example would be that a
movement detection requires at least 5
frames to be detected.
    /// The `minimumLatencyFrameCount`
for that request would report 5 and only
after 5 frames have been processed an
observation would be returned in the
results.
    /// This latency is indicative of how
responsive a request is in respect to the
incoming data.
    var minimumLatencyFrameCount: Int {
get }

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
```

```
processed.
    var frameAnalysisSpacing: CMTime {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension StatefulRequest {

    /// The minimum number of frames that
the request has to process on before
reporting back any observation.
    ///
    /// This information is provided by
the request once initialized with its
required paramters.
    /// Video-based requests often need a
minimum number of frames before they can
report back any observation.
    /// An example would be that a
movement detection requires at least 5
frames to be detected.
    /// The `minimumLatencyFrameCount`
for that request would report 5 and only
after 5 frames have been processed an
observation would be returned in the
results.
    /// This latency is indicative of how
responsive a request is in respect to the
incoming data.
    public var minimumLatencyFrameCount:
Int { 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 == (lhs: Self,
rhs: Self) -> Bool
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension StatefulRequest {

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
```

```
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
TargetedImageRequestHandler : Sendable {

    public convenience init(sourceURL:
URL, targetURL: URL, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(source:
CGImage, target: CGImage, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(source:
CIImage, target: CIImage, orientation:
CGImagePropertyOrientation? = nil)

    public convenience init(source:
```

```
CVPixelBuffer, target: CVPixelBuffer,
orientation: CGImagePropertyOrientation?
= nil)

    public convenience init(source:
CMSampleBuffer, target: CMSampleBuffer,
orientation: CGImagePropertyOrientation?
= nil)

    public convenience init(source: Data,
target: Data, orientation:
CGImagePropertyOrientation? = nil)

    final public func perform<T>(_
request: T) async throws -> T.Result
where T : TargetedRequest

    final public func perform<each T>(_
request: repeat each T) async throws ->
(repeat (each T).Result) where repeat
each T : TargetedRequest

    final public func performAll(_
requests: some Collection<any
TargetedRequest>) -> some
AsyncSequence<VisionResult, Never>

}

/// A request that can be used with a
`TargetedImageHandler` to analyze two
images together.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol TargetedRequest :
VisionRequest {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct TextObservation :
VisionObservation, QuadrilateralProviding
{

    /// An array of detected individual
character bounding boxes.
    ///
    /// If the associated
`DetectTextRectanglesRequest` indicates
interest in character boxes by setting
the option `reportCharacterBoxes` to
`true`,
    /// this property is non-`nil`. If no
characters are found, it remains empty.
    public let characterBoxes:
[RectangleObservation]?

    /// The bounding box of the object.
    ///
    /// The coordinate system is
normalized to the dimensions of the
processed image, with the origin at the
lower-left corner of the image.
    public let boundingBox:
NormalizedRect
```

```
    /// The coordinates of the upper-left
corner of the quadrilateral.
    public let topLeft: NormalizedPoint

    /// The coordinates of the upper-
right corner of the quadrilateral.
    public let topRight: NormalizedPoint

    /// The coordinates of the lower-
right corner of the quadrilateral.
    public let bottomRight:
NormalizedPoint

    /// The coordinates of the lower-left
corner of the quadrilateral.
    public let bottomLeft:
NormalizedPoint

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
```

```
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
TextObservation, b: TextObservation) ->
Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension TextObservation : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
```

```
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension TextObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNTextObservation)
}

/// An image analysis request, as a
stateful request you track over time,
that determines the perspective warp
matrix necessary to align the content of
two images.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
```

```
TrackHomographicImageRegistrationRequest
: ImageProcessingRequest,
StatefulRequest, TargetedRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
ImageHomographicAlignmentObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
TrackHomographicImageRegistrationRequest.
Revision, b:
TrackHomographicImageRegistrationRequest.
Revision) -> 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:
TrackHomographicImageRegistrationRequest.
Revision, b:
TrackHomographicImageRegistrationRequest.
Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
```

```
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
```

```
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
```

```
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
TrackHomographicImageRegistrationRequest.
Revision? = nil, frameAnalysisSpacing:
CMTime? = nil)

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect

    /// The minimum number of frames that
the request has to process on before
reporting back any observation.
    ///
    /// This information is provided by
the request once initialized with its
required paramters.
    /// Video-based requests often need a
```

```
minimum number of frames before they can
report back any observation.
    /// An example would be that a
movement detection requires at least 5
frames to be detected.
    /// The `minimumLatencyFrameCount`
for that request would report 5 and only
after 5 frames have been processed an
observation would be returned in the
results.
    /// This latency is indicative of how
responsive a request is in respect to the
incoming data.
    final public var
minimumLatencyFrameCount: Int { get }

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime

    /// The request's configured
revision.
    final public let revision:
```

```
TrackHomographicImageRegistrationRequest.
Revision

    /// The collection of currently-
supported revisions for
`TrackHomographicImageRegistrationRequest
`.
    public static let supportedRevisions:
[TrackHomographicImageRegistrationRequest
.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
```

```
}

/// An image analysis request that tracks
the movement of a previously identified
object across multiple images or video
frames.
///
/// Use this type of request to track the
bounding boxes around objects previously
identified in an image. Vision attempts
to locate the same object from the input
observation throughout the sequence.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class TrackObjectRequest :
ImageProcessingRequest, StatefulRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
DetectedObjectObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision2

        /// Returns a Boolean value
indicating whether the value of the first
```

```
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
TrackObjectRequest.Revision, b:
TrackObjectRequest.Revision) -> 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:
TrackObjectRequest.Revision, b:
```

```
TrackObjectRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)
```

```
        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
```

```
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - detectedObject: The object to
track
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    ///   - frameAnalysisSpacing: The
duration between analysis operations.
Increase this value to reduce the number
of frames analyzed on slower devices. By
default all frames will be analyzed.
    public init(detectedObject: any
BoundingBoxProviding & VisionObservation,
_ revision: TrackObjectRequest.Revision?
= nil, frameAnalysisSpacing: CMTime? =
nil)
```

```
    /// The object which will be tracked.
    final public let inputObservation:
any BoundingBoxProviding &
VisionObservation

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime

    /// The request's configured
```

```
revision.
    final public let revision:
TrackObjectRequest.Revision

    /// The collection of currently-
supported revisions for
`TrackObjectRequest`.
    public static let supportedRevisions:
[TrackObjectRequest.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
TrackOpticalFlowRequest :
ImageProcessingRequest, StatefulRequest,
TargetedRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
OpticalFlowObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
```

```
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
TrackOpticalFlowRequest.Revision, b:
TrackOpticalFlowRequest.Revision) -> 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:
TrackOpticalFlowRequest.Revision, b:
TrackOpticalFlowRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
```

```
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
```

```
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
```

```
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
TrackOpticalFlowRequest.Revision? = nil,
frameAnalysisSpacing: CMTime? = nil)

    public enum ComputationAccuracy :
CaseIterable, Sendable, Equatable,
Codable, Hashable {

        case low

        case medium

        case high

        case veryHigh

        /// 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:
TrackOpticalFlowRequest.ComputationAccura
cy, b:
TrackOpticalFlowRequest.ComputationAccura
cy) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[TrackOpticalFlowRequest.ComputationAccur
acy]

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[TrackOpticalFlowRequest.ComputationAccur
acy] { get }

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    final public var computationAccuracy:
TrackOpticalFlowRequest.ComputationAccura
cy

    /// The collection of supported pixel
format types.
    final public var
supportedOutputPixelFormatTypes: [OSType]
{ get }

    /// The desired pixel format type of
the observation. The default is
`kCVPixelFormatType_TwoComponent32Float`.
    final public var
outputPixelFormatType: OSType

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
```

```
    final public let
frameAnalysisSpacing: CMTime

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect

    /// The request's configured
revision.
    final public let revision:
TrackOpticalFlowRequest.Revision

    /// The collection of currently-
supported revisions for
`TrackOpticalFlowRequest`.
    public static let supportedRevisions:
[TrackOpticalFlowRequest.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
TrackRectangleRequest :
ImageProcessingRequest, StatefulRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
RectangleObservation?

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
TrackRectangleRequest.Revision, b:
TrackRectangleRequest.Revision) -> 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:
TrackRectangleRequest.Revision, b:
TrackRectangleRequest.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
```

```
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
```

```
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// - Parameters:
    ///   - detectedRectangle: The
rectangle to track
    ///   - revision: The specific
algorithm or implementation revision
that's used to perform the request.
    ///   - frameAnalysisSpacing: The
duration between analysis operations.
Increase this value to reduce the number
of frames analyzed on slower devices. By
```

```
default all frames will be analyzed.
    public init(detectedRectangle: any
QuadrilateralProviding &
VisionObservation, _ revision:
TrackRectangleRequest.Revision? = nil,
frameAnalysisSpacing: CMTime? = nil)

    public enum TrackingLevel :
CaseIterable, Sendable, Equatable,
Codable, Hashable {

        case accurate

        case fast

        /// 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:
TrackRectangleRequest.TrackingLevel, b:
TrackRectangleRequest.TrackingLevel) ->
Bool

        /// Hashes the essential
```

```
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// A type that can represent a
collection of all values of this type.
        @available(iOS 18.0, tvOS 18.0,
visionOS 2.0, macOS 15.0, *)
        public typealias AllCases =
[TrackRectangleRequest.TrackingLevel]

        /// A collection of all values of
this type.
        nonisolated public static var
allCases:
[TrackRectangleRequest.TrackingLevel] {
get }

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
```

```
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    /// The object which will be tracked.
    final public let inputObservation:
any QuadrilateralProviding &
VisionObservation
```

```
    final public var trackingLevel:
TrackRectangleRequest.TrackingLevel

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime

    /// The request's configured
revision.
    final public let revision:
TrackRectangleRequest.Revision
```

```
    public static let supportedRevisions:
[TrackRectangleRequest.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}

/// An image analysis request, as a
stateful request you track over time,
that determines the affine transform
necessary to align the content of two
images.
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class
TrackTranslationalImageRegistrationReques
t : ImageProcessingRequest,
StatefulRequest, TargetedRequest {

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    public typealias Result =
ImageTranslationAlignmentObservation

    public enum Revision : Comparable,
Sendable, Equatable, Codable, Hashable {

        case revision1

        /// Returns a Boolean value
indicating whether the value of the first
        /// argument is less than that of
the second argument.
        ///
        /// This function is the only
requirement of the `Comparable` protocol.
The
        /// remainder of the relational
operator functions are implemented by the
        /// standard library for any type
that conforms to `Comparable`.
```

```
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func < (a:
TrackTranslationalImageRegistrationReques
t.Revision, b:
TrackTranslationalImageRegistrationReques
t.Revision) -> 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:
TrackTranslationalImageRegistrationReques
t.Revision, b:
TrackTranslationalImageRegistrationReques
t.Revision) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// Encodes this value into the
given encoder.
        ///
        /// If the value fails to encode
anything, `encoder` will encode an empty
        /// keyed container in its place.
        ///
        /// This function throws an error
if any values are invalid for the given
        /// encoder's format.
        ///
        /// - Parameter encoder: The
encoder to write data to.
        public func encode(to encoder:
any Encoder) throws

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }

        /// Creates a new instance by
decoding from the given decoder.
        ///
        /// This initializer throws an
```

```
error if reading from the decoder fails,
or
        /// if the data read is corrupted
or otherwise invalid.
        ///
        /// - Parameter decoder: The
decoder to read data from.
        public init(from decoder: any
Decoder) throws
    }

    public init(_ revision:
TrackTranslationalImageRegistrationReques
t.Revision? = nil, frameAnalysisSpacing:
CMTime? = nil)

    final public var
supportedComputeStageDevices:
[ComputeStage : [MLComputeDevice]] {
get }

    /// The region of the image in which
Vision will perform the request.
    ///
    /// The rectangle is normalized to
the dimensions of the processed image.
Its origin is specified relative to the
image's lower-left corner.
    /// By default, the region of
interest will be the full image.
    final public var regionOfInterest:
NormalizedRect
```

```
    /// The minimum number of frames that
the request has to process on before
reporting back any observation.
    ///
    /// This information is provided by
the request once initialized with its
required paramters.
    /// Video-based requests often need a
minimum number of frames before they can
report back any observation.
    /// An example would be that a
movement detection requires at least 5
frames to be detected.
    /// The `minimumLatencyFrameCount`
for that request would report 5 and only
after 5 frames have been processed an
observation would be returned in the
results.
    /// This latency is indicative of how
responsive a request is in respect to the
incoming data.
    final public var
minimumLatencyFrameCount: Int { get }

    /// The reciprocal of maximum rate at
which buffers will be processed.
    ///
    /// The request will not process
buffers that fall within the
`frameAnalysisSpacing` since the
previously performed analysis.
    /// The analysis is not done by wall
time but by analysis of of the time
```

```
stamps of the samplebuffers being
processed.
    final public let
frameAnalysisSpacing: CMTime

    /// The request's configured
revision.
    final public let revision:
TrackTranslationalImageRegistrationReques
t.Revision

    /// The collection of currently-
supported revisions for
`TrackTranslationalImageRegistrationReque
st`.
    public static let supportedRevisions:
[TrackTranslationalImageRegistrationReque
st.Revision]

    /// An enum that identifies the
request and request revision.
    final public var descriptor:
RequestDescriptor { get }

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
```

```
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    final public var hashValue: Int { get
}
}

/// An observation that describes a
detected trajectory.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public struct TrajectoryObservation :
VisionObservation {

    /// The centroid points of the
detected contour along the trajectory.
    ///
    /// The detected points may differ
slightly from the ideal trajectory
because they fall within the allowed
tolerance. The system limits the maximum
number of points based on the maximum
trajectory length set in the request.
    public let detectedPoints:
[NormalizedPoint]

    /// The centroids of the calculated
trajectory from the detected points.
    ///
    /// The projected points define the
```

```
ideal trajectory described by the
parabolic equation. The equation’s
coefficients and the projected points of
the detected trajectory get refined over
time. The system limits the maximum
number of cached points to the maximum
points needed to describe the trajectory
together with the parabolic equation.
    public let projectedPoints:
[NormalizedPoint]

    /// The coefficients of the parabolic
equation.
    ///
    /// This equation describes the
parabola on which the detected contour is
traveling. The equation and the projected
points get refined over time.
    public let equationCoefficients:
simd_float3

    /// The moving average radius of the
object the request is tracking.
    public let movingAverageRadius:
CGFloat

    /// The unique identifier for the
observation.
    public let uuid: UUID

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
```

```
    ///
    /// The only exception is results
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    public let confidence: Float

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    public let timeRange: CMTimeRange?

    /// The descriptor of the request
that produced the observation.
    public let
originatingRequestDescriptor:
RequestDescriptor?

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
```

```
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: 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:
TrajectoryObservation, b:
TrajectoryObservation) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension TrajectoryObservation : Codable
{

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
```

```
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

extension TrajectoryObservation {

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
    public init(_ observation:
VNTrajectoryObservation)
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
final public class VideoProcessor :
Sendable {

    public enum Cadence : Sendable,
Equatable, Hashable {

        case timeInterval(CMTime)

        case frameInterval(Int)

        /// 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:
VideoProcessor.Cadence, b:
VideoProcessor.Cadence) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
        /// your program. Do not save
```

```
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }
    }

    public init(_ videoURL: URL)

    final public func addRequest<T>(_
request: T, cadence:
VideoProcessor.Cadence? = nil) async
throws -> some AsyncSequence<T.Result,
any Error> where T : VisionRequest


    final public func removeRequest(_
request: any VisionRequest) async -> Bool

    final public func cancel() async

    final public func startAnalysis(of
timeRange: CMTimeRange? = nil)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum VisionError : Error {

    /// Required data is missing.
    case dataUnavailable(String)

    public static func
dataUnavailable(message: String) ->
VisionError

    /// An internal error occurred within
the framework.
    case internalError(String)

    public static func
internalError(message: String) ->
VisionError

    /// A request was configured with an
invalid value.
    case invalidArgument(String)

    public static func
invalidArgument(message: String) ->
VisionError

    /// Data is formatted incorrectly.
    case invalidFormat(String)

    public static func
invalidFormat(message: String) ->
VisionError
```

```
    /// The input image is invalid.
    ///
    /// This error occurs when you pass
an invalid image to an operation, such as
passing an image with no dimensions.
    case invalidImage(String)

    public static func
invalidImage(message: String) ->
VisionError

    /// The Core ML model is incompatible
with the request.
    case invalidModel(String)

    public static func
invalidModel(message: String) ->
VisionError

    /// An app requested an unsupported
operation.
    case invalidOperation(String)

    public static func
invalidOperation(message: String) ->
VisionError

    /// An I/O error for an image, image
sequence, or Core ML model.
    case ioError(String)

    public static func ioError(message:
String) -> VisionError
```

```
    /// The requested operation failed.
    case operationFailed(String)

    public static func
operationFailed(message: String) ->
VisionError

    /// An app attempted to access data
that’s out-of-bounds.
    case outOfBoundsError(String)

    public static func
outOfBoundsError(message: String) ->
VisionError

    /// There is not enough memory to
perform the operation.
    case outOfMemory(String)

    public static func
outOfMemory(message: String) ->
VisionError

    /// An error occurred while creating
a pixel buffer.
    case
pixelBufferCreationFailed(CVReturn)

    public static func
pixelBufferCreationFailed(errorCode:
CVReturn) -> VisionError
```

```
    /// An app cancelled the request.
    case requestCancelled(String)

    public static func
requestCancelled(message: String) ->
VisionError

    /// The operation timed out.
    case timeout(String)

    public static func timeout(message:
String) -> VisionError

    /// The system can’t find a
timestamp.
    case timeStampNotFound(String)

    public static func
timeStampNotFound(message: String) ->
VisionError

    /// An app requested an unsupported
compute device.
    case unsupportedComputeDevice(String)

    public static func
unsupportedComputeDevice(message: String)
-> VisionError

    /// An app requested an unsupported
compute stage.
    case unsupportedComputeStage(String)
    public static func
unsupportedComputeStage(message: String)
-> VisionError

    /// An app attempted an unsupported
request.
    case unsupportedRequest(String)

    public static func
unsupportedRequest(message: String) ->
VisionError

    /// An app specified an unsupported
request revision.
    case unsupportedRevision(String)

    public static func
unsupportedRevision(message: String) ->
VisionError

    public static func
espressoE5RTError(message: String,
errorCode: UInt32) -> VisionError

    public static func
unimplementedFunction(message: String) ->
VisionError
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension VisionError : LocalizedError {
    /// A localized message describing
what error occurred.
    public var errorDescription: String?
{ get }
}

extension VisionError {

    public static func requestCancelled(_
request: (any VisionRequest)) ->
VisionError

    public static func
performDisallowedFromMultipleThreads(_
request: (any VisionRequest)) ->
VisionError
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol VisionObservation :
CustomStringConvertible, Decodable,
Encodable, Hashable, Sendable {

    /// The unique identifier for the
observation.
    var uuid: UUID { get }

    /// The level of confidence
normalized to `[0, 1]` where `1` is most
confident.
    ///
    /// The only exception is results
```

```
coming from `CoreMLRequest`, where
confidence values are forwarded as is
from relevant CoreML models
    var confidence: Float { get }

    /// The time range of the reported
observation.
    ///
    /// When evaluating a sequence of
image buffers, use this property to
determine each observation’s start time
and duration.
    var timeRange: CMTimeRange? { get }

    /// The descriptor of the request
that produced the observation.
    var originatingRequestDescriptor:
RequestDescriptor? { get }

    override func hash(into hasher: inout
Hasher)

    override var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension VisionObservation {

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)
}

/// A protocol for Vision analysis
requests.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public protocol VisionRequest :
CustomStringConvertible, Hashable,
Sendable {
    var supportedComputeStageDevices:
[ComputeStage : [MLComputeDevice]] {
get }

    /// Returns the compute device for a
compute stage.
    ///
    /// - Parameters:
    ///  - `computeStage: `The compute
stage to inspect.
    ///
    /// - Returns: The current compute
device; otherwise, `nil` if one isn’t
assigned.
    func computeDevice(for computeStage:
ComputeStage) -> MLComputeDevice?

    /// Assigns a compute device for a
compute stage.
    ///
    /// If the parameter `computeDevice`
is `nil`, the framework removes any
explicit compute device assignment and
allows the framework to select the
device.
    /// Configure any compute device for
a given compute stage. When performing a
request, the system makes a validity
check. Use `supportedComputeStageDevices`
to get valid compute devices for a
request’s compute stages.
    ///
    /// - Parameters:
```

```
    ///  - `computeDevice: `The compute
device to assign to the compute stage.
    ///  - `computeStage: `The compute
stage.
    mutating func setComputeDevice(_
computeDevice: MLComputeDevice?, for
computeStage: ComputeStage)

    /// An enum that identifies the
request and request revision.
    var descriptor: RequestDescriptor {
get }

    /// The type produced by performing a
Request.
    ///
    /// This type will either be a single
VisionObservation or array of
VisionObservations.
    associatedtype Result
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
public enum VisionResult : Sendable {

    case
calculateImageAestheticsScores(CalculateI
mageAestheticsScoresRequest,
ImageAestheticsScoresObservation)

    case
classifyImage(ClassifyImageRequest,
[ClassificationObservation])

    case coreML(CoreMLRequest, [any
VisionObservation])

    case
detectAnimalBodyPose(DetectAnimalBodyPose
Request, [AnimalBodyPoseObservation])

    case
detectBarcodes(DetectBarcodesRequest,
[BarcodeObservation])

    case
detectContours(DetectContoursRequest,
ContoursObservation)

    case
detectDocumentSegmentation(DetectDocument
SegmentationRequest,
DetectedDocumentObservation?)

    case
detectFaceCaptureQuality(DetectFaceCaptur
eQualityRequest, [FaceObservation])

    case
detectFaceLandmarks(DetectFaceLandmarksRe
quest, [FaceObservation])

    case
detectHorizon(DetectHorizonRequest,
HorizonObservation?)
```

```
    case
detectHumanBodyPose(DetectHumanBodyPoseRe
quest, [HumanBodyPoseObservation])

    case
detectHumanBodyPose3D(DetectHumanBodyPose
3DRequest, [HumanBodyPose3DObservation])

    case
detectHumanHandPose(DetectHumanHandPoseRe
quest, [HumanHandPoseObservation])

    case
detectRectangles(DetectRectanglesRequest,
[RectangleObservation])

    case
detectTextRectangles(DetectTextRectangles
Request, [TextObservation])

    case
detectTrajectories(DetectTrajectoriesRequ
est, [TrajectoryObservation])

    case
generateAttentionBasedSaliencyImage(Gener
ateAttentionBasedSaliencyImageRequest,
SaliencyImageObservation)

    case
generateImageFeaturePrint(GenerateImageFe
aturePrintRequest,
FeaturePrintObservation)

    case
generateForegroundInstanceMask(GenerateFo
regroundInstanceMaskRequest,
InstanceMaskObservation?)

    case
generateObjectnessBasedSaliencyImage(Gene
rateObjectnessBasedSaliencyImageRequest,
SaliencyImageObservation)

    case
generatePersonSegmentation(GeneratePerson
SegmentationRequest,
PixelBufferObservation)

    case
generatePersonInstanceMask(GeneratePerson
InstanceMaskRequest,
InstanceMaskObservation?)

    case
recognizeAnimals(RecognizeAnimalsRequest,
[RecognizedObjectObservation])

    case
recognizeText(RecognizeTextRequest,
[RecognizedTextObservation])

    case
trackHomographicImageRegistration(TrackHo
mographicImageRegistrationRequest,
ImageHomographicAlignmentObservation)

    case trackObject(TrackObjectRequest,
DetectedObjectObservation?)

    case
trackOpticalFlow(TrackOpticalFlowRequest,
OpticalFlowObservation?)

    case
trackRectangle(TrackRectangleRequest,
RectangleObservation?)

    case
trackTranslationalImageRegistration(Track
TranslationalImageRegistrationRequest,
ImageTranslationAlignmentObservation)

    case
detectFaceRectangles(DetectFaceRectangles
Request, [FaceObservation])

    case
detectHumanRectangles(DetectHumanRectangl
esRequest, [HumanObservation])

    case error(any VisionRequest, any
Error)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, visionOS 2.0, *)
extension VisionResult : Equatable,
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
```

```
    public var description: 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 == (lhs:
VisionResult, rhs: VisionResult) -> Bool
}

@available(macOS 10.13, iOS 11.0, tvOS
11.0, *)
extension VNFaceLandmarkRegion2D {

    @nonobjc public var normalizedPoints:
[CGPoint] { get }

    @nonobjc public func
pointsInImage(imageSize: CGSize) ->
[CGPoint]

    @available(macOS 10.15, iOS 13.0,
tvOS 13.0, *)
    @nonobjc public var
```

```
precisionEstimatesPerPoint: [Float]? {
get }
}

@available(macOS 10.15, iOS 13.0, tvOS
13.0, *)
extension VNFaceObservation {

    @nonobjc public var
faceCaptureQuality: Float? { get }
}

@available(macOS 10.15, iOS 13.0, tvOS
13.0, *)
extension VNRecognizedText {

    @nonobjc public func boundingBox(for
range: Range<String.Index>) throws ->
VNRectangleObservation?
}

@available(macOS 11.0, iOS 14.0, tvOS
14.0, *)
extension VNContour {

    @nonobjc public var normalizedPoints:
[simd_float2] { get }
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, *)
extension VNDetectHumanBodyPoseRequest {
    @nonobjc public var
supportedJointNames:
[VNHumanBodyPoseObservation.JointName] {
get throws }

    @nonobjc public var
supportedJointsGroupNames:
[VNHumanBodyPoseObservation.JointsGroupNa
me] { get throws }
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, *)
extension VNDetectHumanHandPoseRequest {

    @nonobjc public var
supportedJointNames:
[VNHumanHandPoseObservation.JointName] {
get throws }

    @nonobjc public var
supportedJointsGroupNames:
[VNHumanHandPoseObservation.JointsGroupNa
me] { get throws }
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, *)
extension VNDetectAnimalBodyPoseRequest {

    @nonobjc public var
supportedJointNames:
[VNAnimalBodyPoseObservation.JointName] {
```

```
get throws }

    @nonobjc public var
supportedJointsGroupNames:
[VNAnimalBodyPoseObservation.JointsGroupN
ame] { get throws }
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, *)
extension VNHumanBodyPose3DObservation {

    @nonobjc public func
cameraRelativePosition(_ jointName:
VNHumanBodyPose3DObservation.JointName)
throws -> simd_float4x4
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, *)
extension VNDetectHumanBodyPose3DRequest
{

    @nonobjc public var
supportedJointNames:
[VNHumanBodyPose3DObservation.JointName]
{ get throws }

    @nonobjc public var
supportedJointsGroupNames:
[VNHumanBodyPose3DObservation.JointsGroup
Name] { get throws }
}
```

```
extension VNBarcodeSymbology {

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "aztec")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "aztec")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "aztec")
    @available(visionOS, deprecated: 1.0,
renamed: "aztec")
    public static var Aztec:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "code39")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "code39")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "code39")
    @available(visionOS, deprecated: 1.0,
renamed: "code39")
    public static var Code39:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed:
"code39Checksum")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed:
"code39Checksum")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed:
```

```
"code39Checksum")
    @available(visionOS, deprecated: 1.0,
renamed: "code39Checksum")
    public static var Code39Checksum:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed:
"code39FullASCII")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed:
"code39FullASCII")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed:
"code39FullASCII")
    @available(visionOS, deprecated: 1.0,
renamed: "code39FullASCII")
    public static var Code39FullASCII:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed:
"code39FullASCIIChecksum")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed:
"code39FullASCIIChecksum")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed:
"code39FullASCIIChecksum")
    @available(visionOS, deprecated: 1.0,
renamed: "code39FullASCIIChecksum")
    public static var
Code39FullASCIIChecksum:
```

```
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "code93")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "code93")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "code93")
    @available(visionOS, deprecated: 1.0,
renamed: "code93")
    public static var Code93:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "code93i")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "code93i")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "code93i")
    @available(visionOS, deprecated: 1.0,
renamed: "code93i")
    public static var Code93i:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "code128")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "code128")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "code128")
    @available(visionOS, deprecated: 1.0,
renamed: "code128")
    public static var Code128:
```

```
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "dataMatrix")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "dataMatrix")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "dataMatrix")
    @available(visionOS, deprecated: 1.0,
renamed: "dataMatrix")
    public static var DataMatrix:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "ean8")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "ean8")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "ean8")
    @available(visionOS, deprecated: 1.0,
renamed: "ean8")
    public static var EAN8:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "ean13")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "ean13")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "ean13")
    @available(visionOS, deprecated: 1.0,
renamed: "ean13")
    public static var EAN13:
```

```
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "i2of5")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "i2of5")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "i2of5")
    @available(visionOS, deprecated: 1.0,
renamed: "i2of5")
    public static var I2of5:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed:
"i2of5Checksum")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed:
"i2of5Checksum")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed:
"i2of5Checksum")
    @available(visionOS, deprecated: 1.0,
renamed: "i2of5Checksum")
    public static var I2of5Checksum:
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13,
deprecated: 12.0, renamed: "itf14")
    @available(iOS, introduced: 11.0,
deprecated: 15.0, renamed: "itf14")
    @available(tvOS, introduced: 11.0,
deprecated: 15.0, renamed: "itf14")
```

```
    @available(visionOS, deprecated: 1.0, 
renamed: "itf14")
    public static var ITF14: 
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13, 
deprecated: 12.0, renamed: "pdf417")
    @available(iOS, introduced: 11.0, 
deprecated: 15.0, renamed: "pdf417")
    @available(tvOS, introduced: 11.0, 
deprecated: 15.0, renamed: "pdf417")
    @available(visionOS, deprecated: 1.0, 
renamed: "pdf417")
    public static var PDF417: 
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13, 
deprecated: 12.0, renamed: "qr")
    @available(iOS, introduced: 11.0, 
deprecated: 15.0, renamed: "qr")
    @available(tvOS, introduced: 11.0, 
deprecated: 15.0, renamed: "qr")
    @available(visionOS, deprecated: 1.0, 
renamed: "qr")
    public static var QR: 
VNBarcodeSymbology { get }

    @available(macOS, introduced: 10.13, 
deprecated: 12.0, renamed: "upce")
    @available(iOS, introduced: 11.0, 
deprecated: 15.0, renamed: "upce")
    @available(tvOS, introduced: 11.0, 
deprecated: 15.0, renamed: "upce")
```

```
    @available(visionOS, deprecated: 1.0,
renamed: "upce")
    public static var UPCE:
VNBarcodeSymbology { get }
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, *)
extension VNRequest {

    @nonobjc public var
supportedComputeStageDevices:
[VNComputeStage : [MLComputeDevice]] {
get throws }

    @nonobjc public func
computeDevice(for computeStage:
VNComputeStage) -> MLComputeDevice?

    @nonobjc public func
setComputeDevice(_ computeDevice:
MLComputeDevice?, for computeStage:
VNComputeStage)
}
```