```
import CoreFoundation
import CoreGraphics.CGAffineTransform
import CoreGraphics.CGBase
import CoreGraphics.CGBitmapContext
import CoreGraphics.CGColor
import CoreGraphics.CGColorConversionInfo
import CoreGraphics.CGColorSpace
import CoreGraphics.CGContext
import
CoreGraphics.CGConvertColorDataWithFormat
import CoreGraphics.CGDataConsumer
import CoreGraphics.CGDataProvider
import CoreGraphics.CGDirectDisplay
import CoreGraphics.CGDirectDisplayMetal
import CoreGraphics.CGDirectPalette
import
CoreGraphics.CGDisplayConfiguration
import CoreGraphics.CGDisplayFade
import CoreGraphics.CGDisplayStream
import CoreGraphics.CGEXRToneMappingGamma
import CoreGraphics.CGError
import CoreGraphics.CGEvent
import CoreGraphics.CGEventSource
import CoreGraphics.CGEventTypes
import CoreGraphics.CGFont
import CoreGraphics.CGFunction
import CoreGraphics.CGGeometry
import CoreGraphics.CGGradient
import CoreGraphics.CGITUToneMapping
import CoreGraphics.CGImage
import CoreGraphics.CGLayer
import CoreGraphics.CGPDFArray
import CoreGraphics.CGPDFContentStream
```

```swift
import CoreGraphics.CGPDFContext
import CoreGraphics.CGPDFDictionary
import CoreGraphics.CGPDFDocument
import CoreGraphics.CGPDFObject
import CoreGraphics.CGPDFOperatorTable
import CoreGraphics.CGPDFPage
import CoreGraphics.CGPDFScanner
import CoreGraphics.CGPDFStream
import CoreGraphics.CGPDFString
import CoreGraphics.CGPSConverter
import CoreGraphics.CGPath
import CoreGraphics.CGPattern
import CoreGraphics.CGRemoteOperation
import CoreGraphics.CGSession
import CoreGraphics.CGShading
import CoreGraphics.CGToneMapping
import CoreGraphics.CGWindow
import CoreGraphics.CGWindowLevel
import Foundation
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func CGGetLastMouseDelta() -> (x:
Int32, y: Int32)

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public enum CGPathFillRule : Int {

    /// Nonzero winding number fill rule.
    ///
    /// This rule plots a ray from the
interior of the region to be evaluated
    /// toward the bounds of the drawing,
and sums the closed path elements
    /// that the ray crosses: +1 for
counterclockwise paths, -1 for clockwise.
    /// If the sum is zero, the region is
left empty; if the sum is nonzero,
    /// the region is filled.
    case winding

    /// Even-Odd fill rule.
    ///
    /// This rule plots a ray from the
interior of the region to be evaluated
    /// toward the bounds of the drawing,
and sums the closed path elements
    /// that the ray crosses.
    /// If the sum is an even numner, the
region is left empty; if the sum is
    /// an odd number, the region is
filled.
    case evenOdd

    /// Creates a new instance with the
specified raw value.
    ///
    /// If there is no value of the type
that corresponds with the specified raw
    /// value, this initializer returns
`nil`. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     print(PaperSize(rawValue:
"Legal"))
    ///     // Prints
"Optional("PaperSize.Legal")"
    ///
    ///     print(PaperSize(rawValue:
"Tabloid"))
    ///     // Prints "nil"
    ///
    /// - Parameter rawValue: The raw
value to use for the new instance.
    public init?(rawValue: Int)

    /// The raw type that can be used to
represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
conforming type has a corresponding
unique
    /// value of the `RawValue` type, but
there may be values of the `RawValue`
    /// type that don't have a
corresponding value of the conforming
type.
    @available(iOS 7.0, tvOS 9.0, watchOS
2.0, visionOS 1.0, macOS 10.9, *)
    public typealias RawValue = Int
    /// The corresponding value of the
raw type.
    ///
    /// A new instance initialized with
`rawValue` will be equivalent to this
    /// instance. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     let selectedSize =
PaperSize.Letter
    ///     print(selectedSize.rawValue)
    ///     // Prints "Letter"
    ///
    ///     print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    ///     // Prints "true"
    public var rawValue: Int { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPathFillRule : Equatable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPathFillRule : Hashable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPathFillRule :
RawRepresentable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func acos(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func acosh(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func asin(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func asinh(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func atan(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func atan2(_ lhs: CGFloat, _ rhs:
```

```
CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func atanh(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func cbrt(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func copysign(_ lhs: CGFloat, _
rhs: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func cos(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func cosh(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func erf(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func erfc(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func exp(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func exp2(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func expm1(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func fdim(_ lhs: CGFloat, _ rhs:
CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func fmax(_ lhs: CGFloat, _ rhs:
CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func fmin(_ lhs: CGFloat, _ rhs:
CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func hypot(_ lhs: CGFloat, _ rhs:
CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
@available(swift, deprecated: 4.2,
message: "use the exponent property.")
public func ilogb(_ x: CGFloat) -> Int

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func j0(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func j1(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func jn(_ n: Int, _ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
@available(swift, deprecated: 4.2,
renamed: "scalbn")
public func ldexp(_ x: CGFloat, _ n: Int)
-> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func lgamma(_ x: CGFloat) ->
(CGFloat, Int)

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func log(_ x: CGFloat) -> CGFloat
```

```
@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func log10(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func log1p(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func log2(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func logb(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
@available(swift, deprecated: 4.2,
message: "use CGFloat(nan:
CGFloat.RawSignificand) instead.")
public func nan(_ tag: String) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func nearbyint(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
```

```
public func nextafter(_ lhs: CGFloat, _
rhs: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func pow(_ lhs: CGFloat, _ rhs:
CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func remquo(_ x: CGFloat, _ y:
CGFloat) -> (CGFloat, Int)

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func rint(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func sin(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func sinh(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func tan(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func tanh(_ x: CGFloat) -> CGFloat
```

```
@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func tgamma(_ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func y0(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func y1(_ x: CGFloat) -> CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
public func yn(_ n: Int, _ x: CGFloat) ->
CGFloat

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGColor {

    public var components: [CGFloat]? {
get }

    public class var white: CGColor { get
}

    public class var black: CGColor { get
}

    public class var clear: CGColor { get
}
```

```
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGColor : @unchecked Sendable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPoint {

    public static var zero: CGPoint { get
}

    public init(x: Int, y: Int)

    public init(x: Double, y: Double)

    public init?(dictionaryRepresentation
dict: CFDictionary)
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPoint : CustomReflectable {

    /// The custom mirror for this
instance.
    ///
    /// If this type has value semantics,
the mirror should be unaffected by
    /// subsequent mutations of the
instance.
```

```
    public var customMirror: Mirror { get
}
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPoint {

    /// A custom playground Quick Look
for this instance.
    ///
    /// If this type has value semantics,
the `PlaygroundQuickLook` instance
    /// should be unaffected by
subsequent mutations.
    @available(*, deprecated, message:
"CGPoint.customPlaygroundQuickLook will
be removed in a future Swift version")
    public var customPlaygroundQuickLook:
PlaygroundQuickLook { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPoint :
CustomDebugStringConvertible {

    /// A textual representation of this
instance, suitable for debugging.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
```

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

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPoint : Equatable {
    /// 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: CGPoint,
rhs: CGPoint) -> Bool
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension 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)

    /// 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, watchOS 11.0, visionOS 2.0, *)
extension CGPoint : Hashable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPoint : 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 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGImage {

    public func
copy(maskingColorComponents components:
[CGFloat]) -> CGImage?
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGImage : @unchecked Sendable {
}

@available(macOS 12.0, iOS 15.0, tvOS
15.0, watchOS 8.0, visionOS 1.0, *)
extension CGEvent {

    public var data: CFData? { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect {

    public static var zero: CGRect {
get }
```

```
    public init(x: CGFloat, y: CGFloat,
width: CGFloat, height: CGFloat)

    public init(x: Double, y: Double,
width: Double, height: Double)

    public init(x: Int, y: Int, width:
Int, height: Int)

    public init?(dictionaryRepresentation
dict: CFDictionary)

    public func divided(atDistance:
CGFloat, from fromEdge: CGRectEdge) ->
(slice: CGRect, remainder: CGRect)
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect : CustomReflectable {

    /// The custom mirror for this
instance.
    ///
    /// If this type has value semantics,
the mirror should be unaffected by
    /// subsequent mutations of the
instance.
    public var customMirror: Mirror { get
}
}
```

```
@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect {

    /// A custom playground Quick Look
for this instance.
    ///
    /// If this type has value semantics,
the `PlaygroundQuickLook` instance
    /// should be unaffected by
subsequent mutations.
    @available(*, deprecated, message:
"CGRect.customPlaygroundQuickLook will be
removed in a future Swift version")
    public var customPlaygroundQuickLook:
PlaygroundQuickLook { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect :
CustomDebugStringConvertible {

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

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

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect : Equatable {

    /// 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: CGRect,
rhs: CGRect) -> Bool
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect {

    /// 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, watchOS 11.0, visionOS 2.0, *)
extension CGRect : Hashable {
}
```

```
@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect : 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 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGAffineTransform {

    public static var identity:
CGAffineTransform { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGAffineTransform : 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 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGAffineTransform : Equatable {

    /// 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:
CGAffineTransform, rhs:
CGAffineTransform) -> Bool
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGAffineTransform {

    /// 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, watchOS 11.0, visionOS 2.0, *)
extension CGAffineTransform : Hashable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGVector {

    public static var zero: CGVector {
get }

    public init(dx: Int, dy: Int)

    public init(dx: Double, dy: Double)
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGVector : Equatable {

    /// 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: CGVector,
rhs: CGVector) -> Bool
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGVector {

    /// 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, watchOS 11.0, visionOS 2.0, *)
extension CGVector : Hashable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGVector :
CustomDebugStringConvertible {

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

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGVector : 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 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize {

    public static var zero: CGSize {
get }

    public init(width: Int, height: Int)

    public init(width: Double, height:
Double)

    public init?(dictionaryRepresentation
dict: CFDictionary)
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize : CustomReflectable {

    /// The custom mirror for this
instance.
    ///
    /// If this type has value semantics,
the mirror should be unaffected by
    /// subsequent mutations of the
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize {

    /// A custom playground Quick Look
for this instance.
    ///
    /// If this type has value semantics,
the `PlaygroundQuickLook` instance
    /// should be unaffected by
subsequent mutations.
    @available(*, deprecated, message:
"CGSize.customPlaygroundQuickLook will be
removed in a future Swift version")
    public var customPlaygroundQuickLook:
PlaygroundQuickLook { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize :
CustomDebugStringConvertible {

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

```
in the assignment to `s` uses the
    /// `Point` type's `debugDescription`
property.
    public var debugDescription: String {
get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize : Equatable {

    /// 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: CGSize,
rhs: CGSize) -> Bool
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize {

    /// 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, watchOS 11.0, visionOS 2.0, *)
extension CGSize : Hashable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGSize : 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 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGPath {

    public func copy(dashingWithPhase
phase: CGFloat, lengths: [CGFloat],
transform: CGAffineTransform = .identity)
-> CGPath

    public func copy(strokingWithWidth
lineWidth: CGFloat, lineCap: CGLineCap,
lineJoin: CGLineJoin, miterLimit:
CGFloat, transform: CGAffineTransform
= .identity) -> CGPath

    public func contains(_ point:
CGPoint, using rule: CGPathFillRule
```

```
= .winding, transform: CGAffineTransform
= .identity) -> Bool
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGMutablePath {

    public func addRoundedRect(in rect:
CGRect, cornerWidth: CGFloat,
cornerHeight: CGFloat, transform:
CGAffineTransform = .identity)

    public func move(to point: CGPoint,
transform: CGAffineTransform = .identity)

    public func addLine(to point:
CGPoint, transform: CGAffineTransform
= .identity)

    public func addQuadCurve(to end:
CGPoint, control: CGPoint, transform:
CGAffineTransform = .identity)

    public func addCurve(to end: CGPoint,
control1: CGPoint, control2: CGPoint,
transform: CGAffineTransform = .identity)

    public func addRect(_ rect: CGRect,
transform: CGAffineTransform = .identity)

    public func addRects(_ rects:
[CGRect], transform: CGAffineTransform
```

```
= .identity)

    public func addLines(between points:
[CGPoint], transform: CGAffineTransform =
.identity)

    public func addEllipse(in rect:
CGRect, transform: CGAffineTransform
= .identity)

    public func addRelativeArc(center:
CGPoint, radius: CGFloat, startAngle:
CGFloat, delta: CGFloat, transform:
CGAffineTransform = .identity)

    public func addArc(center: CGPoint,
radius: CGFloat, startAngle: CGFloat,
endAngle: CGFloat, clockwise: Bool,
transform: CGAffineTransform = .identity)

    public func addArc(tangent1End:
CGPoint, tangent2End: CGPoint, radius:
CGFloat, transform: CGAffineTransform
= .identity)

    public func addPath(_ path: CGPath,
transform: CGAffineTransform = .identity)
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension CGPath {
    /// Returns a new path with filled
regions in either this path or the given
path.
    /// - Parameters:
    ///   - other: The path to union.
    ///   - rule: The rule for
determining which areas to treat as the
interior of the paths.
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: A new path.
    ///
    /// The filled region of resulting
path is the combination of the filled
region of both paths added together.
    ///
    /// Any unclosed subpaths in either
path are assumed to be closed. The result
of filling this
    /// path using either even-odd or
non-zero fill rules is identical.
    public func union(_ other: CGPath,
using rule: CGPathFillRule = .winding) ->
CGPath

    /// Returns a new path with filled
regions common to both paths.
    /// - Parameters:
    ///   - other: The path to intersect.
    ///   - rule: The rule for
determining which areas to treat as the
interior of the paths.
```

```
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: A new path.
    ///
    /// The filled region of the
resulting path is the overlapping area of
the filled region of both paths.
    /// This can be used to clip the fill
of a path to a mask.
    ///
    /// Any unclosed subpaths in either
path are assumed to be closed. The result
of filling this
    /// path using either even-odd or
non-zero fill rules is identical.
    public func intersection(_ other:
CGPath, using rule: CGPathFillRule
= .winding) -> CGPath

    /// Returns a new path with filled
regions from this path that are not in
the given path.
    /// - Parameters:
    ///   - other: The path to subtract.
    ///   - rule: The rule for
determining which areas to treat as the
interior of the paths.
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: A new path.
    ///
    /// The filled region of the
resulting path is the filled region of
this path with the filled
    /// region `other` removed from it.
    ///
    /// Any unclosed subpaths in either
path are assumed to be closed. The result
of filling this
    /// path using either even-odd or
non-zero fill rules is identical.
    public func subtracting(_ other:
CGPath, using rule: CGPathFillRule
= .winding) -> CGPath

    /// Returns a new path with filled
regions either from this path or the
given path, but not in both.
    /// - Parameters:
    ///   - other: The path to
difference.
    ///   - rule: The rule for
determining which areas to treat as the
interior of the paths.
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: A new path.
    ///
    /// The filled region of the
resulting path is the filled region
contained in either this path
    /// or `other`, but not both.
    ///
    /// Any unclosed subpaths in either
path are assumed to be closed. The result
of filling this
    /// path using either even-odd or
non-zero fill rules is identical.
    public func symmetricDifference(_
other: CGPath, using rule: CGPathFillRule
= .winding) -> CGPath

    /// Returns a new path with a line
from this path that does not overlap the
filled region of the given path.
    /// - Parameters:
    ///   - other: The path to subtract.
    ///   - rule: The rule for
determining which areas to treat as the
interior of `other`.
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: A new path.
    ///
    /// The line of the resulting path is
the line of this path that does not
overlap the filled region of `other`.
    ///
    /// Intersected subpaths that are
clipped create open subpaths. Closed
subpaths that do not
    /// intersect `other` remain closed.
    public func lineSubtracting(_ other:
CGPath, using rule: CGPathFillRule
= .winding) -> CGPath
```

```
    /// Returns a new path with a line
from this path that overlaps the filled
regions of the given path.
    /// - Parameters:
    ///   - other: The path to intersect.
    ///   - rule: The rule for
determining which areas to treat as the
interior of `other`.
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: A new path.
    ///
    /// The line of the resulting path is
the line of this path that overlaps the
filled region of `other`.
    ///
    /// Intersected subpaths that are
clipped create open subpaths. Closed
subpaths that do not
    /// intersect `other` remain closed.
    public func lineIntersection(_ other:
CGPath, using rule: CGPathFillRule
= .winding) -> CGPath

    /// Returns a new weakly-simple copy
of this path.
    /// - Parameter rule: The rule for
determining which areas to treat as the
interior of the path.
    ///   Defaults to the
`CGPathFillRule.winding` rule if not
```

```
specified.
    /// - Returns: A new path.
    ///
    /// The returned path is weakly-
simple path, has no self-intersections,
and has a normalized
    /// orientation. The result of
filling this path using either even-odd
or non-zero fill rules
    /// is identical.
    public func normalized(using rule:
CGPathFillRule = .winding) -> CGPath

    /// Returns a flattened copy of this
path.
    /// - Parameter threshold: The
maximum error tolerance.
    /// - Returns: A new path.
    ///
    /// The granularity of the
approximation is controlled by
`threshold` the maximum error
    /// tolerance (measured in points)
for curves.
    public func flattened(threshold:
CGFloat) -> CGPath

    /// Returns whether paths overlap.
    /// - Parameters:
    ///   - other: The path to check for
intersection.
    ///   - rule: The rule for
determining which areas to treat as the
```

```
interior of the paths.
    ///     Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: `true` if the paths
intersect.
    ///
    /// This is the same as testing if
the intersection of two paths is not
empty. That is, the
    /// filled areas of the paths
overlap. Open subpaths are treated as
closed subpaths, the same
    /// as when filling a path in a
`CGContext`.
    public func intersects(_ other:
CGPath, using rule: CGPathFillRule
= .winding) -> Bool

    /// Return an array of the visually
separated components of a path.
    /// - Parameter rule: The rule for
determining which areas to treat as the
interior of the path.
    ///   Defaults to the
`CGPathFillRule.winding` rule if not
specified.
    /// - Returns: The visually separate
components.
    ///
    /// Separating components of a path
returns the components from splitting the
path into multiple
```

```
    /// distinct paths. Rendered
individually the components look the same
as the original path.
    /// This can be used to break a
single path (eg. a symbol) into its
component pieces so that each
    /// component can be colored
separately.
    public func componentsSeparated(using
rule: CGPathFillRule = .winding) ->
[CGPath]
}

extension CGAffineTransform {

    public init(translationX tx: CGFloat,
y ty: CGFloat)

    public init(scaleX sx: CGFloat, y sy:
CGFloat)

    public init(rotationAngle angle:
CGFloat)

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public init(_ components:
CGAffineTransformComponents)

    public var isIdentity: Bool { get }

    public func translatedBy(x tx:
CGFloat, y ty: CGFloat) ->
```

```swift
CGAffineTransform

    public func scaledBy(x sx: CGFloat, y
sy: CGFloat) -> CGAffineTransform

    public func rotated(by angle:
CGFloat) -> CGAffineTransform

    public func inverted() ->
CGAffineTransform

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public func decomposed() ->
CGAffineTransformComponents

    public func concatenating(_ t2:
CGAffineTransform) -> CGAffineTransform
}

extension CGAffineTransformComponents {

    @available(macOS 13.0, iOS 16.0,
watchOS 9.0, tvOS 16.0, *)
    public init(scale: CGSize
= .init(width: 1, height: 1),
horizontalShear: CGFloat = 0, rotation:
CGFloat = 0, translation: CGVector
= .zero)
}

extension CGPoint {
    public func applying(_ t:
CGAffineTransform) -> CGPoint
}

extension CGSize {

    public func applying(_ t:
CGAffineTransform) -> CGSize
}

extension CGRect {

    public func applying(_ t:
CGAffineTransform) -> CGRect
}

/// * Definitions of inline functions. **
extension CGAffineTransform {

    public init(_ a: CGFloat, _ b:
CGFloat, _ c: CGFloat, _ d: CGFloat, _
tx: CGFloat, _ ty: CGFloat)
}

extension CGPoint {

    public func equalTo(_ point2:
CGPoint) -> Bool

    /*** Persistent representations. ***/
    public var dictionaryRepresentation:
CFDictionary { get }
}
```

```
extension CGSize {

    public func equalTo(_ size2: CGSize)
-> Bool

    public var dictionaryRepresentation:
CFDictionary { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGRect {

    public static var null: CGRect {
get }

    public static var infinite: CGRect {
get }

    public var minX: CGFloat { get }

    public var midX: CGFloat { get }

    public var maxX: CGFloat { get }

    public var minY: CGFloat { get }

    public var midY: CGFloat { get }

    public var maxY: CGFloat { get }

    public var width: CGFloat { get }
    public var height: CGFloat { get }

    public func equalTo(_ rect2: CGRect)
-> Bool

    public var standardized: CGRect { get
}

    public var isEmpty: Bool { get }

    public var isNull: Bool { get }

    public var isInfinite: Bool { get }

    public func insetBy(dx: CGFloat, dy:
CGFloat) -> CGRect

    public var integral: CGRect { get }

    public func union(_ r2: CGRect) ->
CGRect

    public func intersection(_ r2:
CGRect) -> CGRect

    public func offsetBy(dx: CGFloat, dy:
CGFloat) -> CGRect

    public func contains(_ point:
CGPoint) -> Bool

    public func contains(_ rect2: CGRect)
```

```
-> Bool

    public func intersects(_ rect2:
CGRect) -> Bool

    public var dictionaryRepresentation:
CFDictionary { get }
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGColorSpace {

    public var colorTable: [UInt8]? { get
}
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGColorSpace : @unchecked
Sendable {
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGContext {

    public func setLineDash(phase:
CGFloat, lengths: [CGFloat])

    public func move(to point: CGPoint)

    public func addLine(to point:
```

```
CGPoint)

    public func addCurve(to end: CGPoint,
control1: CGPoint, control2: CGPoint)

    public func addQuadCurve(to end:
CGPoint, control: CGPoint)

    public func addRects(_ rects:
[CGRect])

    public func addLines(between points:
[CGPoint])

    public func addArc(center: CGPoint,
radius: CGFloat, startAngle: CGFloat,
endAngle: CGFloat, clockwise: Bool)

    public func addArc(tangent1End:
CGPoint, tangent2End: CGPoint, radius:
CGFloat)

    /// Fills the current path using the
specified rule (winding by default).
    ///
    /// Any open subpath is implicitly
closed.
    public func fillPath(using rule:
CGPathFillRule = .winding)

    /// Intersects the current path with
the current clipping region and uses the
    /// result as the new clipping region
```

```
for subsequent drawing.
    ///
    /// Uses the specified fill rule
(winding by default) to determine which
    /// areas to treat as the interior of
the clipping region. When evaluating
    /// the path, any open subpath is
implicitly closed.
    public func clip(using rule:
CGPathFillRule = .winding)

    public func fill(_ rects: [CGRect])

    public func
strokeLineSegments(between points:
[CGPoint])

    public func clip(to rects: [CGRect])

    public func draw(_ image: CGImage, in
rect: CGRect, byTiling: Bool = false)

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func draw(_ image: CGImage, in
rect: CGRect, by toneMapping:
CGToneMapping = .default, options:
CFDictionary?) -> Bool

    public var textPosition: CGPoint

    public func showGlyphs(_ glyphs:
[CGGlyph], at positions: [CGPoint])
```

```
}

@available(macOS 10.9, iOS 7.0, tvOS 9.0,
watchOS 2.0, visionOS 1.0, *)
extension CGContext {

    public func draw(_ layer: CGLayer, in
rect: CGRect)

    public func draw(_ layer: CGLayer, at
point: CGPoint)
}
```