```
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims
import os.activity
import os.atomic
import os.lock
import os.log
import os.log
import os.object
import os.overflow
import os.signpost
import os.signpost
import os.trace
import os.trace_base
import os.workgroup
import os.workgroup

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public enum AnimationFormatString {

    /// A type that can be instantiated
with a string literal (format string),
which would
    /// be appended with an animation
tag.
    @frozen public struct OSLogMessage :
ExpressibleByStringLiteral {

        /// Given a string literal,
concatenates it with an animation tag.
        public init(stringLiteral value:
String)
```

```
        /// A type that represents an
extended grapheme cluster literal.
        ///
        /// Valid types for
`ExtendedGraphemeClusterLiteralType` are
`Character`,
        /// `String`, and `StaticString`.
        @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
        public typealias
ExtendedGraphemeClusterLiteralType =
String

        /// A type that represents a
string literal.
        ///
        /// Valid types for
`StringLiteralType` are `String` and
`StaticString`.
        @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
        public typealias
StringLiteralType = String

        /// A type that represents a
Unicode scalar literal.
        ///
        /// Valid types for
`UnicodeScalarLiteralType` are
`Unicode.Scalar`,
        /// `Character`, `String`, and
`StaticString`.
```

```
        @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
        public typealias
UnicodeScalarLiteralType = String
    }
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension
AnimationFormatString.OSLogMessage :
BitwiseCopyable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public struct Logger : @unchecked
Sendable {

    /// Creates a custom logger for
logging to a specific subsystem and
category.
    public init(subsystem: String,
category: String)

    /// Creates a logger for logging to
the default subsystem.
    public init()

    /// Creates a logger instance from an
existing OSLog object that has the
subsystem and
    /// category.
    public init(_ logObj: OSLog)

    /// Logs a string interpolation at
the `default` level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.log("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
```

```
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.log("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.log("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func log(_ message:
OSLogMessage)

    /// Logs a string interpolation at
the specified log level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     let errorLevel = .fault
    ///     logger.log(level: errorLevel,
"A string interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.log(
    ///       level: .debug,
    ///       "An unsigned integer \(x,
format: .hex, align: .right(columns:
10))")
    ///
    ///     logger.log(
    ///       level: errorLevel,
    ///       "A unsigned integer \(x,
privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameters:
    ///   - level: Logging level.
    ///   - message: A string
interpolation.
    public func log(level: OSLogType, _
message: OSLogMessage)

    /// An alias for `debug`. Logs a
string interpolation at the debug level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.trace("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
```

```
Expressions and Specifying Privacy
    ///
========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.trace("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.trace("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func trace(_ message:
OSLogMessage)

    /// Logs a string interpolation at
the `debug` level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
```

```
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.debug("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.debug("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
```

```
    ///     logger.debug("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func debug(_ message:
OSLogMessage)

    /// Logs a string interpolation at
the `info` level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.info("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.info("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.info("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func info(_ message:
OSLogMessage)

    /// Logs a string interpolation at
the `default` level.
    ///
    /// Values that can be interpolated
```

```
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.notice("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.notice("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.notice("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func notice(_ message:
OSLogMessage)

    /// An alias for `error`. Logs a
string interpolation at the `error`
level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.warning("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.warning("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.warning("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func warning(_ message:
OSLogMessage)

    /// Logs a string interpolation at
the `error` level.
```

```
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.error("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.error("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.error("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func error(_ message:
OSLogMessage)

    /// Logs a string interpolation at
the most severe level: `fault`.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.critical("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.critical("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.critical("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func critical(_ message:
OSLogMessage)

    /// Logs a string interpolation at
```

```
the `fault` level.
    ///
    /// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
    /// Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let logger = Logger()
    ///     logger.fault("A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     logger.fault("An unsigned
integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     logger.fault("An unsigned
integer \(x, privacy: .private)")
    ///
    /// - Warning: Do not explicity
create OSLogMessage. Instead pass a
string interpolation.
    ///
    /// - Parameter message: A string
interpolation.
    public func fault(_ message:
OSLogMessage)
}

/// An `OSAllocatedUnfairLock` is a
wrapper around an `os_unfair_lock` that
locks
/// around accesses to a stored object.
///
/// In Swift, `os_unfair_lock` is unsafe
to use directly with `&` because, as
/// a value type, its instances do not
have stable addresses. This wrapper
avoids
/// that pitfall — despite being a
`struct`, it isn't a value type, as
copied
/// instances control the same underlying
lock allocation.
///
```

```
/// Prefer storing state protected by the
lock in `State`. Containing locked state
/// inside the lock helps track what is
protected state and provides a scope
/// where it is safe to access that
state.
///
/// When using OSAllocatedUnfairLock with
external state, nonscoped locking
/// allows more flexible locking patterns
by using `lock()` / `unlock()`, but
/// offers no assistance in tracking what
state is protected by the lock.
///
/// This lock must be unlocked from the
same thread that locked it.  As such, it
/// is unsafe to use `lock()` /
`unlock()` across an `await` suspension
point.
/// Instead, use `withLock` to enforce
that the lock is only held within
/// a synchronous scope.
///
/// If you are using a lock from
asynchronous contexts only,
/// prefer using an actor instead.
///
/// This lock is not a recursive lock.
Attempting to lock it again from the same
/// thread while the lock is already
locked will crash.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
```

```
@frozen public struct
OSAllocatedUnfairLock<State> : @unchecked
Sendable {

    /// Initialize an
OSAllocatedUnfairLock with a non-sendable
lock-protected
    /// `initialState`.
    ///
    /// By initializing with a non-
sendable type, the owner of this
structure
    /// must ensure the Sendable contract
is upheld manually.
    /// Non-sendable content from `State`
should not be allowed
    /// to escape from the lock.
    ///
    /// - Parameter initialState: An
initial value to store that will be
    ///  protected under the lock.
    ///
    public init(uncheckedState
initialState: State)

    ///  Perform a closure while holding
this lock.
    ///  This method does not enforce
sendability requirement
    ///  on closure body and its return
type.
    ///  The caller of this method is
responsible for ensuring references
```

```
    ///   to non-sendables from closure
uphold the Sendability contract.
    ///
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: The return value of
`body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func withLockUnchecked<R>(_
body: (inout State) throws -> R) rethrows
-> R

    ///  Perform a closure while holding
this lock.
    ///  This method does not enforce
sendability requirement
    ///  on closure body and its return
type.
    ///  The caller of this method is
responsible for ensuring references
    ///   to non-sendables from closure
uphold the Sendability contract.
    ///
    /// - Parameter flags: Flags to alter
the behavior of the lock. See
OSAllocatedUnfairLock.Flags.
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: The return value of
`body`.
    /// - Throws: Anything thrown by
```

```
`body`.
    ///
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func
withLockUnchecked<R>(flags:
OSAllocatedUnfairLockFlags, _ body:
(inout State) throws -> R) rethrows -> R

    ///  Perform a sendable closure while
holding this lock.
    ///
    ///
    /// - Parameter body: A sendable
closure to invoke while holding this
lock.
    /// - Returns: The sendable return
value of `body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func withLock<R>(_ body:
@Sendable (inout State) throws -> R)
rethrows -> R where R : Sendable

    ///  Perform a sendable closure while
holding this lock.
    ///
    ///
    /// - Parameter flags: Flags to alter
the behavior of the lock. See
OSAllocatedUnfairLock.Flags.
    /// - Parameter body: A sendable
```

```
closure to invoke while holding this
lock.
    /// - Returns: The sendable return
value of `body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func withLock<R>(flags:
OSAllocatedUnfairLockFlags, _ body:
@Sendable (inout State) throws -> R)
rethrows -> R where R : Sendable

    ///  Attempt to acquire the lock, if
successful, perform a closure while
    ///  holding the lock.
    ///  This method does not enforce
sendability requirement
    ///  on closure body and its return
type.
    ///  The caller of this method is
responsible for ensuring references
    ///   to non-sendables from closure
uphold the Sendability contract.
    ///
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: If the lock is
acquired, the result of `body`.
    ///            If the lock is not
acquired, nil.
    /// - Throws: Anything thrown by
```

```
`body`.
    ///
    public func
withLockIfAvailableUnchecked<R>(_ body:
(inout State) throws -> R) rethrows -> R?

    ///  Attempt to acquire the lock, if
successful, perform a sendable closure
while
    ///  holding the lock.
    ///
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: If the lock is
acquired, the result of `body`.
    ///           If the lock is not
acquired, nil.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func withLockIfAvailable<R>(_
body: @Sendable (inout State) throws ->
R) rethrows -> R? where R : Sendable

    /// Represent ownership status for
`precondition` checking.
    @frozen public enum Ownership {

        /// Lock is currently owned by
the calling thread.
        case owner

        /// Lock is unlocked or owned by
```

```
a different thread.
        case notOwner

        /// 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:
OSAllocatedUnfairLock<State>.Ownership,
b:
OSAllocatedUnfairLock<State>.Ownership)
-> 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 }
    }

    /// Check a precondition about
whether the calling thread is the lock
owner.
    ///
    /// - Parameter condition: An
`Ownership` statement to check for the
    /// current context.
    /// - If the lock is currently owned
by the calling thread:
    ///   - `.owner` - returns
    ///   - `.notOwner` - asserts and
terminates the process
    /// - If the lock is unlocked or
owned by a different thread:
    ///   - `.owner` - asserts and
terminates the process
    ///   - `.notOwner` - returns
    ///
    public func precondition(_ condition:
OSAllocatedUnfairLock<State>.Ownership)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension OSAllocatedUnfairLock where
State == () {
    /// Initialize an
OSAllocatedUnfairLock with no protected
state.
    public init()

    ///  Perform a closure while holding
this lock.
    ///  This method does not enforce
sendability requirement
    ///  on closure body and its return
type.
    ///  The caller of this method is
responsible for ensuring references
    ///   to non-sendables from closure
uphold the Sendability contract.
    ///
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: The return value of
`body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func withLockUnchecked<R>(_
body: () throws -> R) rethrows -> R

    ///  Perform a closure while holding
this lock.
    ///  This method does not enforce
sendability requirement
    ///  on closure body and its return
type.
    ///  The caller of this method is
```

```
responsible for ensuring references
    ///   to non-sendables from closure
uphold the Sendability contract.
    ///
    /// - Parameter flags: Flags to alter
the behavior of the lock. See
OSAllocatedUnfairLock.Flags.
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: The return value of
`body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func
withLockUnchecked<R>(flags:
OSAllocatedUnfairLockFlags, _ body: ()
throws -> R) rethrows -> R

    ///  Perform a sendable closure while
holding this lock.
    ///
    /// - Parameter body: A sendable
closure to invoke while holding this
lock.
    /// - Returns: The return value of
`body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func withLock<R>(_ body:
```

```
@Sendable () throws -> R) rethrows -> R
where R : Sendable

    ///  Perform a sendable closure while
holding this lock.
    ///
    /// - Parameter flags: Flags to alter
the behavior of the lock. See
OSAllocatedUnfairLock.Flags.
    /// - Parameter body: A sendable
closure to invoke while holding this
lock.
    /// - Returns: The return value of
`body`.
    /// - Throws: Anything thrown by
`body`.
    ///
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func withLock<R>(flags:
OSAllocatedUnfairLockFlags, _ body:
@Sendable () throws -> R) rethrows -> R
where R : Sendable

    ///  Attempt to acquire the lock, if
successful, perform a closure while
    ///  holding the lock.
    ///  This method does not enforce
sendability requirement
    ///  on closure body and its return
type.
    ///  The caller of this method is
responsible for ensuring references
```

```
    ///   to non-sendables from closure
uphold the Sendability contract.
    ///
    /// - Parameter body: A closure to
invoke while holding this lock.
    /// - Returns: If the lock is
acquired, the result of `body`.
    ///            If the lock is not
acquired, nil.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func
withLockIfAvailableUnchecked<R>(_ body:
() throws -> R) rethrows -> R?

    ///  Attempt to acquire the lock, if
successful, perform a sendable closure
while
    ///  holding the lock.
    ///
    /// - Parameter body: A sendable
closure to invoke while holding this
lock.
    /// - Returns: If the lock is
acquired, the result of `body`.
    ///            If the lock is not
acquired, nil.
    /// - Throws: Anything thrown by
`body`.
    ///
    public func withLockIfAvailable<R>(_
body: @Sendable () throws -> R) rethrows
```

```
-> R? where R : Sendable

    /// Acquire this lock.
    public func lock()

    /// Acquire this lock.
    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public func lock(flags:
OSAllocatedUnfairLockFlags)

    /// Unlock this lock.
    public func unlock()

    /// Attempt to acquire the lock if it
is not already locked.
    ///
    /// - Returns: `true` if the lock was
succesfully locked, and
    ///  `false` if the lock attempt
failed.
    public func lockIfAvailable() -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension OSAllocatedUnfairLock where
State : Sendable {

    /// Initialize an
OSAllocatedUnfairLock with a lock-
protected sendable
    /// `initialState`.
    /// - Parameter initialState: An
initial value to store that will be
    ///   protected under the lock.
    public init(initialState: State)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension OSAllocatedUnfairLock.Ownership
: Equatable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension OSAllocatedUnfairLock.Ownership
: Hashable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension OSAllocatedUnfairLock.Ownership
: Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension OSAllocatedUnfairLock.Ownership
: BitwiseCopyable {
}

/// `OSAllocatedUnfairLockFlags` provides
flags to alter the behavior of the
/// `OSAllocatedUnfairLock's` lock APIs.`
```

```
///
/// OSAllocatedUnfairLock struct handles
generics types and to avoid nesting a
/// a specific type like
OSAllocatedUnfairLockFlags inside it, it
has been
/// implemented as a separate struct.
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct
OSAllocatedUnfairLockFlags : OptionSet,
RawRepresentable {

    /// 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 let rawValue: UInt32

    /// Creates a new option set from the
given raw value.
    ///
    /// This initializer always succeeds,
even if the value passed as `rawValue`
    /// exceeds the static properties
declared as part of the option set. This
    /// example creates an instance of
`ShippingOptions` with a raw value beyond
    /// the highest element, with a bit
mask that effectively contains all the
    /// declared static members.
    ///
    ///     let extraOptions =
ShippingOptions(rawValue: 255)
    ///
print(extraOptions.isStrictSuperset(of: .
all))
    ///     // Prints "true"
    ///
    /// - Parameter rawValue: The raw
value of the option set to create. Each
bit
    ///   of `rawValue` potentially
represents an element of the option set,
    ///   though raw values may include
bits that are not defined as distinct
    ///   values of the `OptionSet` type.
    public init(rawValue: UInt32)
    ///
    /// This flag allows the caller of
OSAllocatedUnfairLock lock APIs to spin
temporarily
    /// before blocking, particularly
useful when the holder of the lock is on
core.
    /// This should only be used for
locks where the protected critical
section is always
    /// extremely short.
    public static let adaptiveSpin:
OSAllocatedUnfairLockFlags

    /// The type of the elements of an
array literal.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias ArrayLiteralElement
= OSAllocatedUnfairLockFlags

    /// The element type of the option
set.
    ///
    /// To inherit all the default
implementations from the `OptionSet`
protocol,
    /// the `Element` type must be
`Self`, the default.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
```

```
    public typealias Element =
OSAllocatedUnfairLockFlags

    /// 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,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias RawValue = UInt32
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public enum OSLogBoolFormat {

    /// Displays an interpolated boolean
value as true or false.
    case truth

    /// Displays an interpolated boolean
value as yes or no.
    case answer
    /// 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:
OSLogBoolFormat, b: OSLogBoolFormat) ->
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 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogBoolFormat : Equatable {
}
```

```
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogBoolFormat : Hashable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@frozen public struct
OSLogFloatFormatting {

    /// Displays an interpolated
floating-point value in fprintf's `%f`
format with
    /// default precision.
    ///
    /// Prints all digits before the
radix point, and 6 digits following the
radix point.
    /// Note also that this format is
very likely to print non-zero values as
all-zero.
    ///
    /// Note that very large floating-
point values may print quite a lot of
digits
    /// when using this format --up to
hundreds for `Double`. Note also that
this
    /// format is very likely to print
non-zero values as all-zero. If these are
a concern,
    /// use `.exponential` or `.hybrid`
instead.
```

```
    @inlinable public static var fixed:
OSLogFloatFormatting { get }

    /// Displays an interpolated
floating-point value in fprintf's `%f`
format with
    /// specified precision, and optional
sign and case.
    ///
    /// Prints all digits before the
radix point, and `precision` digits
following
    /// the radix point. If `precision`
is zero, the radix point is omitted.
    ///
    /// Note that very large floating-
point values may print quite a lot of
digits
    /// when using this format, even if
`precision` is zero--up to hundreds for
    /// `Double`. Note also that this
format is very likely to print non-zero
values as
    /// all-zero. If these are a concern,
use `.exponential` or `.hybrid` instead.
    ///
    /// Systems may impose an upper bound
on the number of digits that are
    /// supported following the radix
point.
    ///
    /// All parameters to this function
except `precision` must be boolean
```

```
literals.
    ///
    /// - Parameters:
    ///   - precision: Number of digits
to display after the radix point.
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
fixed(precision: @autoclosure @escaping
() -> Int, explicitPositiveSign: Bool =
false, uppercase: Bool = false) ->
OSLogFloatFormatting

    /// Displays an interpolated
floating-point value in fprintf's `%f`
format with
    /// default precision, and optional
sign and case.
    ///
    /// Prints all digits before the
radix point, and 6 digits following the
radix point.
    /// Note also that this format is
very likely to print non-zero values as
all-zero.
    ///
    /// Note that very large floating-
point values may print quite a lot of
```

```
digits
    /// when using this format, even if
`precision` is zero--up to hundreds for
    /// `Double`. Note also that this
format is very likely to print non-zero
values as
    /// all-zero. If these are a concern,
use `.exponential` or `.hybrid` instead.
    ///
    /// Systems may impose an upper bound
on the number of digits that are
    /// supported following the radix
point.
    ///
    /// All parameters to this function
must be boolean literals.
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
fixed(explicitPositiveSign: Bool = false,
uppercase: Bool = false) ->
OSLogFloatFormatting

    /// Displays an interpolated
floating-point value in hexadecimal
format.
    @inlinable public static var hex:
```

```
OSLogFloatFormatting { get }

    /// Displays an interpolated
floating-point value in hexadecimal
format with
    /// optional sign and case.
    ///
    /// All parameters to this function
must be boolean literals.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
hex(explicitPositiveSign: Bool = false,
uppercase: Bool = false) ->
OSLogFloatFormatting

    /// Displays an interpolated
floating-point value in fprintf's `%e`
format.
    ///
    /// Prints the number in the form
[-]d.ddd...dde±dd.
    @inlinable public static var
exponential: OSLogFloatFormatting { get }

    /// Displays an interpolated
```

```
floating-point value in fprintf's `%e`
format with
    /// specified precision, and optional
sign and case.
    ///
    /// Prints the number in the form
[-]d.ddd...dde±dd, with `precision`
significant
    /// digits following the radix point.
Systems may impose an upper bound on the
number
    /// of digits that are supported.
    ///
    /// All parameters except `precision`
must be boolean literals.
    ///
    /// - Parameters:
    ///   - precision: Number of digits
to display after the radix point.
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
exponential(precision: @autoclosure
@escaping () -> Int,
explicitPositiveSign: Bool = false,
uppercase: Bool = false) ->
OSLogFloatFormatting
```

```
    /// Displays an interpolated
floating-point value in fprintf's `%e`
format with
    /// an optional sign and case.
    ///
    /// Prints the number in the form
[-]d.ddd...dde±dd.
    ///
    /// All parameters to this function
must be boolean literals.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
exponential(explicitPositiveSign: Bool =
false, uppercase: Bool = false) ->
OSLogFloatFormatting

    /// Displays an interpolated
floating-point value in fprintf's `%g`
format.
    ///
    /// Behaves like `.fixed` when the
number is scaled close to 1.0, and like
    /// `.exponential` if it has a very
large or small exponent.
    @inlinable public static var hybrid:
```

```
OSLogFloatFormatting { get }

    /// Displays an interpolated
floating-point value in fprintf's `%g`
format with the
    /// specified precision, and optional
sign and case.
    ///
    /// Behaves like `.fixed` when the
number is scaled close to 1.0, and like
    /// `.exponential` if it has a very
large or small exponent.
    ///
    /// All parameters except `precision`
must be boolean literals.
    ///
    /// - Parameters:
    ///   - precision: Number of digits
to display after the radix point.
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
hybrid(precision: @autoclosure @escaping
() -> Int, explicitPositiveSign: Bool =
false, uppercase: Bool = false) ->
OSLogFloatFormatting

    /// Displays an interpolated
```

```
floating-point value in fprintf's `%g`
format with
    /// optional sign and case.
    ///
    /// Behaves like `.fixed` when the
number is scaled close to 1.0, and like
    /// `.exponential` if it has a very
large or small exponent.
    ///
    /// All parameters to this function
must be boolean literals.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - uppercase: Pass `true` to use
uppercase letters or `false` to use
    ///     lowercase letters. The
default is `false`.
    @inlinable public static func
hybrid(explicitPositiveSign: Bool =
false, uppercase: Bool = false) ->
OSLogFloatFormatting
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogFloatFormatting.Notation :
Equatable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogFloatFormatting.Notation :
Hashable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public enum OSLogInt32ExtendedFormat {

    /// Displays an interpolated Int32
value as IPv4 address.
    /// For instance, 0x0100007f would be
displayed as 127.0.0.1
    case ipv4Address

    /// Displays an interpolated Int32
value as seconds elapsed since  00:00:00
UTC on 1 January 1970.
    case secondsSince1970

    /// Displays an interpolated Int32
value as Darwin errno. For example: 32:
Broken pipe.
    case darwinErrno

    /// Displays an interpolated Int32
value as Darwin file mode. For example:
-rwxrwxrwx.
    case darwinMode

    /// Displays an interpolated Int32
value as Darwin signal. For example:
sigsegv: Segmentation fault.
```

```
    case darwinSignal

    /// Displays an interpolated Int32
value as Mach errno. For example: 0x6:
(os/kern) resource shortage
    case machErrno

    /// Displays an interpolated Int32
value as bitrate. For example: 100 kbps.
    case bitrate

    /// Displays an interpolated Int32
value as IEC bitrate. For example: 1
Gibps.
    case bitrateIEC

    /// Displays an interpolated Int32
value as bytes. For example: 1 MB.
    @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
    case byteCount

    /// Displays an interpolated Int32
value as IEC bytes. For example: 1 MiB.
    @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
    case byteCountIEC

    /// Displays an interpolated Int32
value as true or false depending
    /// on whether it is non-zero or
zero. It is a short hand
for .boolean(.truth).
```

```
    case truth

    /// Displays an interpolated Int32
value as yes or no depending
    /// on whether it is non-zero or
zero. It is a short hand
for .boolean(.answer).
    case answer

    /// 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:
OSLogInt32ExtendedFormat, b:
OSLogInt32ExtendedFormat) -> 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 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInt32ExtendedFormat :
Equatable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInt32ExtendedFormat :
Hashable {
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public enum OSLogIntExtendedFormat {

    /// Displays an interpolated Int
value as bitrate. For example: 100 kbps.
    case bitrate

    /// Displays an interpolated Int
value as IEC bitrate. For example: 1
Gibps.
    case bitrateIEC

    /// Displays an interpolated Int
value as bytes. For example: 1 MB.
    case byteCount
    /// Displays an interpolated Int
value as IEC bytes. For example: 1 MiB.
    case byteCountIEC

    /// 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:
OSLogIntExtendedFormat, b:
OSLogIntExtendedFormat) -> 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 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension OSLogIntExtendedFormat :
Equatable {
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
extension OSLogIntExtendedFormat :
Hashable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@frozen public struct
OSLogIntegerFormatting {

    /// Displays an interpolated integer
as a decimal number with the specified
number
    /// of digits and an optional sign.
    ///
    /// The parameter
`explicitPositiveSign` must be a boolean
literal. The
    /// parameter `minDigits` can be an
arbitrary expression.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - minDigits: minimum number of
digits to display. Numbers will be
```

```
    ///     prefixed with zeros if
necessary to meet the minimum.
    @inlinable public static func
decimal(explicitPositiveSign: Bool =
false, minDigits: @autoclosure @escaping
() -> Int) -> OSLogIntegerFormatting

    /// Displays an interpolated integer
as a decimal number with an optional
sign.
    ///
    /// The parameter
`explicitPositiveSign` must be a boolean
literal.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    @inlinable public static func
decimal(explicitPositiveSign: Bool =
false) -> OSLogIntegerFormatting

    /// Displays an interpolated integer
as a decimal number. This is the default
format for
    /// integers.
    @inlinable public static var decimal:
OSLogIntegerFormatting { get }

    /// Displays an interpolated unsigned
integer as a hexadecimal number with the
    /// specified parameters. This
```

```
formatting option should be used only
with unsigned
    /// integers.
    ///
    /// All parameters except `minDigits`
should be boolean literals. `minDigits`
    /// can be an arbitrary expression.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - includePrefix: Pass `true` to
add a prefix 0x.
    ///   - uppercase: Pass `true` to use
uppercase letters to represent numerals
    ///     greater than 9, or `false` to
use lowercase letters. The default is
`false`.
    ///   - minDigits: minimum number of
digits to display. Numbers will be
    ///     prefixed with zeros if
necessary to meet the minimum.
    @inlinable public static func
hex(explicitPositiveSign: Bool = false,
includePrefix: Bool = false, uppercase:
Bool = false, minDigits: @autoclosure
@escaping () -> Int) ->
OSLogIntegerFormatting

    /// Displays an interpolated unsigned
integer as a hexadecimal number with the
specified
```

```
    /// parameters. This formatting
option should be used only with unsigned
integers.
    ///
    /// All parameters  should be boolean
literals.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - includePrefix: Pass `true` to
add a prefix 0x.
    ///   - uppercase: Pass `true` to use
uppercase letters to represent numerals
    ///     greater than 9, or `false` to
use lowercase letters. The default is
`false`.
    @inlinable public static func
hex(explicitPositiveSign: Bool = false,
includePrefix: Bool = false, uppercase:
Bool = false) -> OSLogIntegerFormatting

    /// Displays an interpolated unsigned
integer as a hexadecimal number.
    /// This formatting option should be
used only with unsigned integers.
    @inlinable public static var hex:
OSLogIntegerFormatting { get }

    /// Displays an interpolated unsigned
integer as an octal number with the
specified
```

```
    /// parameters. This formatting
option should be used only with unsigned
    /// integers.
    ///
    /// All parameters except `minDigits`
should be boolean literals. `minDigits`
    /// can be an arbitrary expression.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - includePrefix: Pass `true` to
add a prefix 0o.
    ///   - uppercase: Pass `true` to use
uppercase letters to represent numerals
    ///     greater than 9, or `false` to
use lowercase letters. The default is
`false`.
    ///   - minDigits: minimum number of
digits to display. Numbers will be
    ///     prefixed with zeros if
necessary to meet the minimum.
    @inlinable public static func
octal(explicitPositiveSign: Bool = false,
includePrefix: Bool = false, uppercase:
Bool = false, minDigits: @autoclosure
@escaping () -> Int) ->
OSLogIntegerFormatting

    /// Displays an interpolated unsigned
integer as an octal number with the
specified parameters.
```

```
    /// This formatting option should be
used only with unsigned integers.
    ///
    /// All parameters must be boolean
literals.
    ///
    /// - Parameters:
    ///   - explicitPositiveSign: Pass
`true` to add a + sign to non-negative
    ///     numbers.
    ///   - includePrefix: Pass `true` to
add a prefix 0o.
    ///   - uppercase: Pass `true` to use
uppercase letters to represent numerals
    ///     greater than 9, or `false` to
use lowercase letters.
    @inlinable public static func
octal(explicitPositiveSign: Bool = false,
includePrefix: Bool = false, uppercase:
Bool = false) -> OSLogIntegerFormatting

    /// Displays an interpolated unsigned
integer as an octal number.
    /// This formatting option should be
used only with unsigned integers.
    @inlinable public static var octal:
OSLogIntegerFormatting { get }
}

/// Represents a string interpolation
passed to the log APIs.
///
/// This type converts (through its
methods) the given string interpolation
into
/// a C-style format string and a
sequence of arguments.
///
/// - Warning: Do not explicitly refer to
this type. It will be implicitly created
/// by the compiler when you pass a
string interpolation to the log APIs.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@frozen public struct
OSLogInterpolation :
StringInterpolationProtocol {

    /// Creates an empty instance ready
to be filled with string literal content.
    ///
    /// Don't call this initializer
directly. Instead, initialize a variable
or
    /// constant using a string literal
with interpolated expressions.
    ///
    /// Swift passes this initializer a
pair of arguments specifying the size of
    /// the literal segments and the
number of interpolated segments. Use this
    /// information to estimate the
amount of storage you will need.
    ///
    /// - Parameter literalCapacity: The
approximate size of all literal segments
```

```
    ///   combined. This is meant to be
passed to `String.reserveCapacity(_:)`;
    ///   it may be slightly larger or
smaller than the sum of the counts of
each
    ///   literal segment.
    /// - Parameter interpolationCount:
The number of interpolations which will
be
    ///   appended. Use this value to
estimate how much additional capacity
will
    ///   be needed for the interpolated
segments.
    @inlinable public
init(literalCapacity: Int,
interpolationCount: Int)

    /// Appends a literal segment to the
interpolation.
    ///
    /// Don't call this method directly.
Instead, initialize a variable or
    /// constant using a string literal
with interpolated expressions.
    ///
    /// Interpolated expressions don't
pass through this method; instead, Swift
    /// selects an overload of
`appendInterpolation`. For more
information, see
    /// the top-level
`StringInterpolationProtocol`
```

```
documentation.
    ///
    /// - Parameter literal: A string
literal containing the characters
    ///   that appear next in the string
literal.
    @inlinable public mutating func
appendLiteral(_ literal: String)

    /// The type that should be used for
literal segments.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias StringLiteralType =
String
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for
expressions of type NSObject.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `NSObject` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - argumentObject: The
```

```
interpolated expression of type NSObject,
which is autoclosured.
    ///   - privacy: A privacy qualifier
which is either private or public. It is
auto-inferred by default.
    @inlinable public mutating func
appendInterpolation(_ argumentObject:
@autoclosure @escaping () -> NSObject,
privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for
expressions of type NSObject.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `NSObject` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - argumentObject: The
interpolated expression of type NSObject,
which is autoclosured.
    ///   - privacy: A privacy qualifier
which is either private or public. It is
auto-inferred by default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
```

```
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ argumentObject:
@autoclosure @escaping () -> NSObject,
privacy: OSLogPrivacy = .auto,
attributes: String)

    public mutating func
appendInterpolation(_ object:
@autoclosure @escaping () -> NSObject?,
privacy: OSLogPrivacy = .auto,
attributes: String = "")
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for
UnsafeRawBufferPointer.
    ///
    /// Do not call this function
directly.
    ///
    /// - Parameters:
```

```
    ///   - pointer: The interpolated
expression of type
`UnsafeRawBufferPointer`, which is
autoclosured.
    ///   - format: An instance of
`OSLogPointerFormat`. The default value
is .none.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ pointer:
@autoclosure @escaping () ->
UnsafeRawBufferPointer, format:
OSLogPointerFormat = .none, privacy:
OSLogPrivacy = .auto)

    /// Defines interpolation for
UnsafeRawBufferPointer.
    ///
    /// Do not call this function
directly.
    ///
    /// - Parameters:
    ///   - pointer: The interpolated
expression of type
`UnsafeRawBufferPointer`, which is
autoclosured.
    ///   - format: An instance of
`OSLogPointerFormat`. The default value
is .none.
    ///   - privacy: A privacy qualifier
```

```
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ pointer:
@autoclosure @escaping () ->
UnsafeRawBufferPointer, format:
OSLogPointerFormat = .none, privacy:
OSLogPrivacy = .auto, attributes: String)

    /// Defines interpolation for
UnsafeRawPointer.
    ///
    /// Do not call this function
directly.
    ///
    /// - Parameters:
    ///   - pointer: The interpolated
```

```
expression of type `UnsafeRawPointer`,
which is autoclosured.
    ///   - bytes: The size of the
pointee in bytes.
    ///   - format: An instance of
`OSLogPointerFormat`. The default value
is .none.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ pointer:
@autoclosure @escaping () ->
UnsafeRawPointer, bytes: @autoclosure
@escaping () -> Int, format:
OSLogPointerFormat = .none, privacy:
OSLogPrivacy = .auto)

    /// Defines interpolation for
UnsafeRawPointer.
    ///
    /// Do not call this function
directly.
    ///
    /// - Parameters:
    ///   - pointer: The interpolated
expression of type `UnsafeRawPointer`,
which is autoclosured.
    ///   - bytes: The size of the
pointee in bytes.
    ///   - format: An instance of
`OSLogPointerFormat`. The default value
```

```
is .none.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ pointer:
@autoclosure @escaping () ->
UnsafeRawPointer, bytes: @autoclosure
@escaping () -> Int, format:
OSLogPointerFormat = .none, privacy:
OSLogPrivacy = .auto, attributes: String)
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for
expressions of type Int.
```

```
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Int` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type Int, which is
autoclosured.
    ///   - format: A formatting option
available for integer types, defined by
the
    ///     type:
`OSLogIntegerFormatting`. The default is
`.decimal`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int, format:
OSLogIntegerFormatting = .decimal, align:
OSLogStringAlignment = .none, privacy:
OSLogPrivacy = .auto)
```

```
    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int8,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int16,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int32,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int64,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for
expressions of type UInt.
    ///
    /// Do not call this function
```

```
directly. It will be called automatically
when interpolating
    /// a value of type `Int` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type UInt, which is
autoclosured.
    ///   - format: A formatting option
available for integer types, defined by
the
    ///     type
`OSLogIntegerFormatting`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> UInt,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
```

```
@autoclosure @escaping () -> UInt8,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> UInt16,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> UInt32,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> UInt64,
format: OSLogIntegerFormatting
= .decimal, align: OSLogStringAlignment =
.none, privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for
expressions of type Int.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Int` in the
```

```
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type Int, which is
autoclosured.
    ///   - format: A formatting option
available for integer types, defined by
the
    ///     type:
`OSLogIntegerFormatting`. The default is
`.decimal`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
```

```
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation<T>(_ number:
@autoclosure @escaping () -> T, format:
OSLogIntegerFormatting = .decimal, align:
OSLogStringAlignment = .none, privacy:
OSLogPrivacy = .auto, attributes: String)
where T : FixedWidthInteger
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for
expressions of type Float.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Float` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type Float, which is
autoclosured.
    ///   - format: A formatting option
available for float types, defined by the
```

```
    ///     type`OSLogFloatFormatting`.
The default is `.fixed`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Float,
format: OSLogFloatFormatting = .fixed,
align: OSLogStringAlignment = .none,
privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for
expressions of type Float.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Float` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type Float, which is
autoclosured.
    ///   - format: A formatting option
```

```
available for float types, defined by the
    ///     type`OSLogFloatFormatting`.
The default is `.fixed`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Float,
format: OSLogFloatFormatting = .fixed,
align: OSLogStringAlignment = .none,
privacy: OSLogPrivacy = .auto,
attributes: String)

    /// Define interpolation for
expressions of type Double.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Double` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type Double, which is
autoclosured.
    ///   - format: A formatting option
available for float types, defined by the
    ///     type`OSLogFloatFormatting`.
The default is `.fixed`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Double,
format: OSLogFloatFormatting = .fixed,
align: OSLogStringAlignment = .none,
privacy: OSLogPrivacy = .auto)

    /// Define interpolation for
expressions of type Double.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Double` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type Double, which is
autoclosured.
    ///   - format: A formatting option
available for float types, defined by the
    ///     type`OSLogFloatFormatting`.
The default is `.fixed`.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Double,
format: OSLogFloatFormatting = .fixed,
align: OSLogStringAlignment = .none,
privacy: OSLogPrivacy = .auto,
attributes: String)
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    public mutating func
appendInterpolation(_ error: @autoclosure
@escaping () -> any Error, privacy:
OSLogPrivacy = .auto, attributes: String
= "")

    public mutating func
appendInterpolation(_ error: @autoclosure
@escaping () -> (any Error)?, privacy:
OSLogPrivacy = .auto, attributes: String
= "")
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for
expressions of type String.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `String` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - argumentString: The
interpolated expression of type String,
which is autoclosured.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
```

```
appendInterpolation(_ argumentString:
@autoclosure @escaping () -> String,
align: OSLogStringAlignment = .none,
privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for
expressions of type String.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `String` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - argumentString: The
interpolated expression of type String,
which is autoclosured.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
```

```
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ argumentString:
@autoclosure @escaping () -> String,
align: OSLogStringAlignment = .none,
privacy: OSLogPrivacy = .auto,
attributes: String)
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for Int32
expressions that enables formatting them
with os_log
    /// specific formatting options.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Int32` in the
string interpolations passed to the log
APIs.
```

```
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type `Int32`, which is
autoclosured.
    ///   - format: An instance of
`OSLogInt32ExtendedFormat`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int32,
format: OSLogInt32ExtendedFormat,
privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for Int32
expressions that enables formatting them
with os_log
    /// specific formatting options.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Int32` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type `Int32`, which is
autoclosured.
```

```
    ///   - format: An instance of
`OSLogInt32ExtendedFormat`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int32,
format: OSLogInt32ExtendedFormat,
privacy: OSLogPrivacy = .auto,
attributes: String)

    /// Define interpolation for boolean
expressions.
    ///
    /// Do not call this function
directly. It will be called automatically
```

```
when interpolating
    /// a value of type `Bool` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - boolean: The interpolated
expression of type `Bool`, which is
autoclosured.
    ///   - format: An instance of
`OSLogBoolFormat`. Default is `truth`
which implies
    ///     true/false
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///    It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ boolean:
@autoclosure @escaping () -> Bool,
format: OSLogBoolFormat = .truth,
privacy: OSLogPrivacy = .auto)

    /// Defines interpolation for Int
expressions that enables formatting them
with os_log
    /// specific formatting options.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Int` in the
string interpolations passed to the log
```

```
APIs.
    ///
    /// - Parameters:
    ///   - number: The interpolated
expression of type `Int`, which is
autoclosured.
    ///   - format: An instance of
`OSLogIntExtendedFormat`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
    public mutating func
appendInterpolation(_ number:
@autoclosure @escaping () -> Int, format:
OSLogIntExtendedFormat, privacy:
```

```
OSLogPrivacy = .auto, attributes: String
= "")
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation {

    /// Defines interpolation for values
conforming to CustomStringConvertible.
The values
    /// are displayed using the
description methods on them.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value conforming to
CustomStringConvertible in the string
interpolations passed
    /// to the log APIs.
    ///
    /// - Parameters:
    ///   - value: The interpolated
expression conforming to
CustomStringConvertible.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
```

```
    ///     It is auto-inferred by
default.
    public mutating func
appendInterpolation<T>(_ value:
@autoclosure @escaping () -> T, align:
OSLogStringAlignment = .none, privacy:
OSLogPrivacy = .auto) where T :
CustomStringConvertible

    /// Defines interpolation for values
conforming to CustomStringConvertible.
The values
    /// are displayed using the
description methods on them.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value conforming to
CustomStringConvertible in the string
interpolations passed
    /// to the log APIs.
    ///
    /// - Parameters:
    ///   - value: The interpolated
expression conforming to
CustomStringConvertible.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
```

```
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation<T>(_ value:
@autoclosure @escaping () -> T, align:
OSLogStringAlignment = .none, privacy:
OSLogPrivacy = .auto, attributes: String)
where T : CustomStringConvertible

    /// Defines interpolation for meta-
types.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Any.Type` in the
string interpolations passed to the log
```

```
APIs.
    ///
    /// - Parameters:
    ///   - value: An interpolated
expression of type Any.Type, which is
autoclosured.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    @inlinable public mutating func
appendInterpolation(_ value: @autoclosure
@escaping () -> any Any.Type, align:
OSLogStringAlignment = .none, privacy:
OSLogPrivacy = .auto)

    /// Defines interpolation for meta-
types.
    ///
    /// Do not call this function
directly. It will be called automatically
when interpolating
    /// a value of type `Any.Type` in the
string interpolations passed to the log
APIs.
    ///
    /// - Parameters:
    ///   - value: An interpolated
```

```
expression of type Any.Type, which is
autoclosured.
    ///   - align: Left or right
alignment with the minimum number of
columns as
    ///     defined by the type
`OSLogStringAlignment`.
    ///   - privacy: A privacy qualifier
which is either private or public.
    ///     It is auto-inferred by
default.
    ///   - attributes: A string that
specifies an attribute for the
interpolated value,
    ///     which can be used to provide
additional information about the
interpolated
    ///     value to tools such as Xcode
that can process and render os_log and
os_signpost
    ///     messages. An example of an
attribute is "xcode:size-in-bytes". If
the target tool
    ///     that processes these messages
doesn't understand the attribute it would
be ignored.
    public mutating func
appendInterpolation(_ value: @autoclosure
@escaping () -> any Any.Type, align:
OSLogStringAlignment = .none, privacy:
OSLogPrivacy = .auto, attributes: String)
}
```

```
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation.ArgumentType
: Equatable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogInterpolation.ArgumentType
: Hashable {
}

/// Represents a message passed to the
log APIs. This type should be created
/// from a string interpolation or a
string literal.
///
/// Do not explicitly refer to this type.
It will be implicitly created
/// by the compiler when you pass a
string interpolation to the log APIs.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@frozen public struct OSLogMessage :
ExpressibleByStringInterpolation,
ExpressibleByStringLiteral {

    public let interpolation:
OSLogInterpolation

    /// Creates an instance from a string
interpolation.
    ///
    /// Most `StringInterpolation` types
will store information about the
    /// literals and interpolations
appended to them in one or more
properties.
    /// `init(stringInterpolation:)`
should use these properties to initialize
    /// the instance.
    ///
    /// - Parameter stringInterpolation:
An instance of `StringInterpolation`
    ///             which has had each
segment of the string literal appended
    ///             to it.
    @inlinable public
init(stringInterpolation:
OSLogInterpolation)

    /// Creates an instance initialized
to the given string value.
    ///
    /// - Parameter value: The value of
the new instance.
    @inlinable public init(stringLiteral
value: String)

    /// The byte size of the buffer that
will be passed to the logging system.
    @inlinable public var bufferSize: Int
{ get }

    /// A type that represents an
extended grapheme cluster literal.
```

```
    ///
    /// Valid types for
`ExtendedGraphemeClusterLiteralType` are
`Character`,
    /// `String`, and `StaticString`.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias
ExtendedGraphemeClusterLiteralType =
String

    /// The type each segment of a string
literal containing interpolations
    /// should be appended to.
    ///
    /// The `StringLiteralType` of an
interpolation type must match the
    /// `StringLiteralType` of the
conforming type.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias StringInterpolation
= OSLogInterpolation

    /// A type that represents a string
literal.
    ///
    /// Valid types for
`StringLiteralType` are `String` and
`StaticString`.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias StringLiteralType =
String

    /// A type that represents a Unicode
scalar literal.
    ///
    /// Valid types for
`UnicodeScalarLiteralType` are
`Unicode.Scalar`,
    /// `Character`, `String`, and
`StaticString`.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias
UnicodeScalarLiteralType = String
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public enum OSLogPointerFormat {

    /// Pretty prints an `in6_addr`
pointer.
    /// Should only be used with a
pointer typed interpolated expression.
    case ipv6Address

    /// Pretty prints a `timeval`
pointer.
    /// Should only be used with a
pointer typed interpolated expression.
    case timeval

    /// Pretty prints a `timespec`
```

```
pointer.
    /// Should only be used with a
pointer typed interpolated expression.
    case timespec

    /// Pretty prints an `uuid_t`
pointer.
    /// Should only be used with a
pointer typed interpolated expression.
    case uuid

    /// Pretty prints a `sockaddr`
pointer.
    /// Should only be used with a
pointer typed interpolated expression.
    case sockaddr

    /// Displays the raw bytes pointed to
by the pointer.
    case none

    /// 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:
OSLogPointerFormat, b:
OSLogPointerFormat) -> 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 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogPointerFormat : Equatable
{
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogPointerFormat : Hashable {
}

/// Privacy options for specifying
privacy level of the interpolated
expressions
/// in the string interpolations passed
to the log APIs.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@frozen public struct OSLogPrivacy {

    public enum Mask {

        /// Applies a salted hashing
transformation to an interpolated value
to redact it in the logs.
        ///
        /// Its purpose is to permit the
correlation of identical values across
multiple log lines
        /// without revealing the value
itself.
        case hash

        /// No mask
        case none

        /// email: A person name
        /// `mailname`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailName

        /// email: An email address
"tim@apple.com"
        /// `mailaddr`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailAddress

        /// email: The subject of an
```

```
email, e.g. "Beer Bash" or "Re: Beer
Bash"
        /// `mailsubj`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailSubject

        /// email: The "summary" of an
email (usually first (few) lines), e.g.
"Hello there,"
        /// `mailsumm`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailSummary

        /// email: Account name e.g.
"iCloud"
        /// `mailacco`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailAccount

        /// email: Mailbox / mail folder
name e.g. "Some Folder"
        /// `mailbox\0`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailbox

        /// email: Mailbox URL path e.g.
"Marzipan/xfunc"
        /// `mailmbup`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailboxPath

        /// email: Attachment file name
e.g. "News.pdf"
        /// `mailatta`
        @available(macOS 12.0, iOS 15.0,
watchOS 8.0, tvOS 15.0, *)
        case _mailAttachmentFileName

        /// 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:
OSLogPrivacy.Mask, b: OSLogPrivacy.Mask)
-> 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 }
    }

    /// Sets the privacy level of an
interpolated value to public.
    ///
    /// When the privacy level is public,
the value will be displayed
    /// normally without any redaction in
the logs.
    @inlinable public static var
`public`: OSLogPrivacy { get }

    /// Sets the privacy level of an
interpolated value to private.
    ///
    /// When the privacy level is
private, the value will be redacted in
the logs,
    /// subject to the privacy
configuration of the logging system.
    @inlinable public static var
`private`: OSLogPrivacy { get }

    /// Sets the privacy level of an
```

```
interpolated value to private and
    /// applies a `mask` to the
interpolated value to redacted it.
    ///
    /// When the privacy level is
private, the value will be redacted in
the logs,
    /// subject to the privacy
configuration of the logging system.
    ///
    /// If the value need not be redacted
in the logs, its full value is captured
as normal.
    /// Otherwise (i.e. if the value
would be redacted) the `mask` is applied
to
    /// the argument value and the result
of the transformation is recorded
instead.
    ///
    /// - Parameters:
    ///   - mask: Mask to use with the
privacy option.
    @inlinable public static func
`private`(mask: OSLogPrivacy.Mask) ->
OSLogPrivacy

    /// Sets the privacy level of an
interpolated value to sensitive.
    ///
    /// When the privacy level is
sensitive, the value will be redacted in
the logs,
    /// subject to the privacy
configuration of the logging system.
    @inlinable public static var
sensitive: OSLogPrivacy { get }

    /// Sets the privacy level of an
interpolated value to sensitive and
    /// applies a `mask` to the
interpolated value to redacted it.
    ///
    /// When the privacy level is
sensitive, the value will be redacted in
the logs,
    /// subject to the privacy
configuration of the logging system.
    ///
    /// If the value need not be redacted
in the logs, its full value is captured
as normal.
    /// Otherwise (i.e. if the value
would be redacted) the `mask` is applied
to
    /// the argument value and the result
of the transformation is recorded
instead.
    ///
    /// - Parameters:
    ///   - mask: Mask to use with the
privacy option.
    @inlinable public static func
sensitive(mask: OSLogPrivacy.Mask) ->
OSLogPrivacy
```

```
    /// Auto-infers a privacy level for
an interpolated value.
    ///
    /// The system will automatically
decide whether the value should
    /// be captured fully in the logs or
should be redacted.
    @inlinable public static var auto:
OSLogPrivacy { get }

    /// Auto-infers a privacy level for
an interpolated value and applies a
`mask`
    /// to the interpolated value to
redacted it when necessary.
    ///
    /// The system will automatically
decide whether the value should
    /// be captured fully in the logs or
should be redacted.
    /// If the value need not be redacted
in the logs, its full value is captured
as normal.
    /// Otherwise (i.e. if the value
would be redacted) the `mask` is applied
to
    /// the argument value and the result
of the transformation is recorded
instead.
    ///
    /// - Parameters:
    ///   - mask: Mask to use with the
privacy option.
```

```
    @inlinable public static func
auto(mask: OSLogPrivacy.Mask) ->
OSLogPrivacy
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogPrivacy.PrivacyOption :
Equatable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogPrivacy.PrivacyOption :
Hashable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogPrivacy.Mask : Equatable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSLogPrivacy.Mask : Hashable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@frozen public struct
OSLogStringAlignment {

    /// Indicates no alignment.
    @inlinable public static var none:
OSLogStringAlignment { get }

    /// Right align and display at least
`columns` characters.
    ///
    /// The interpolated value would be
padded with spaces, if necessary, to
    /// meet the specified `columns`
characters.
    ///
    /// - Parameter columns: minimum
number of characters to display.
    @inlinable public static func
right(columns: @autoclosure @escaping ()
-> Int) -> OSLogStringAlignment

    /// Left align and display at least
`columns` characters.
    ///
    /// The interpolated value would be
padded with spaces, if necessary, to
    /// meet the specified `columns`
characters.
    ///
    /// - Parameter columns: minimum
number of characters to display.
    @inlinable public static func
left(columns: @autoclosure @escaping ()
-> Int) -> OSLogStringAlignment
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public enum OSSignpostAnimationBegin {

    case animationBegin

    /// 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:
OSSignpostAnimationBegin, b:
OSSignpostAnimationBegin) -> 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 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSSignpostAnimationBegin :
Equatable {
}

@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
extension OSSignpostAnimationBegin :
Hashable {
}

@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
public struct OSSignpostID : Sendable {

    public let rawValue: os_signpost_id_t

    public static let exclusive:
OSSignpostID

    public static let invalid:
OSSignpostID

    public static let null: OSSignpostID

    public init(log: OSLog)

    public init(log: OSLog, object:
AnyObject)

    public init(_ value: UInt64)
}
```

```
@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
extension OSSignpostID : Comparable {

    /// 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:
OSSignpostID, b: OSSignpostID) -> 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:
OSSignpostID, b: OSSignpostID) -> Bool
}

/// A type that tracks the state of an
interval. The state is used in runtime
sanity checks.
@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public class OSSignpostIntervalState :
Codable {

    /// Make a `OSSignpostIntervalState`
from a `OSSignpostID` without calling
`beginInterval`.
    /// Use this function to create an
interval state if the state returned by
the call to
    /// `beginInterval` cannot be passed
to the `endInterval`.
    ///
    /// - Warning: using this to create
an interval state will bypass many
runtime
    /// checks that check for consistency
between beginInterval and endInterval.
    @inlinable public static func
beginState(id: OSSignpostID) ->
OSSignpostIntervalState
```

```
    /// Given a decoder instance, try to
deserialize the proper fields into a
valid
    /// OSSignpostIntervalState instance.
    required public init(from decoder:
any Decoder) throws

    /// Given an encoder, try to
serialize all CodingKeys fields into an
encoding container.
    public func encode(to encoder: any
Encoder) throws
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public struct OSSignposter : @unchecked
Sendable {

    /// Checks if the OSSignposter can
emit signposts.
    public var isEnabled: Bool { get }

    /// A disabled OSSignposter that
won't emit signposts at runtime.
    /// Use to turn off all signposts
emitted using a specific signposter
variable.
    public static var disabled:
OSSignposter { get }

    /// Creates a custom OSSignposter for
```

```
emitting to a specific subsystem and
category.
    public init(subsystem: String,
category: String)

    /// Creates a custom OSSignposter for
emitting to a specific subsystem and
OSLog.category.
    public init(subsystem: String,
category: OSLog.Category)

    /// Creates an OSSignposter for
emitting to the default subsystem.
    public init()

    /// Creates an OSSignposter instance
from an existing OSLog object that has
the subsystem
    /// and category.
    public init(logHandle: OSLog)

    /// Creates an OSSignposter instance
from an existing Logger object that has
the subsystem
    /// and category.
    public init(logger: Logger)

    /// Emits an event signpost to mark a
point of interest in time.
    ///
    /// Values that can be interpolated
in the message parameter include signed
and unsigned Swift integers,
    /// Floats, Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///
signposter.emitEvent("Example", "A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///
signposter.emitEvent("Example", "An
unsigned integer \(x, format: .hex,
align: .right(columns: 10))")
    ///
signposter.emitEvent("Example", "An
unsigned integer \(x,
privacy: .private)")
    ///
    /// - Warning: Do not explicity
create SignpostMetadata. Instead pass a
string interpolation.
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    ///   - message: A string
interpolation that represents the message
you want to add to the signposts.
    public func emitEvent(_ name:
StaticString, id: OSSignpostID
= .exclusive, _ message:
SignpostMetadata)

    /// Emits an event signpost to mark a
point of interest in time.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///
signposter.emitEvent("Example", id:
signpostID)
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    public func emitEvent(_ name:
StaticString, id: OSSignpostID
= .exclusive)

    /// Begins a signposted interval and
returns an open
`OSSignpostIntervalState`.
    ///
    /// Values that can be interpolated
in the message parameter include signed
and unsigned Swift integers,
    /// Floats, Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
```

```
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///     let intervalState =
signposter.beginInterval("Example", id:
signpostID, "A string interpolation \
(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     let intervalState =
signposter.beginInterval("Example", "An
unsigned integer \(x, format: .hex,
align: .right(columns: 10))")
    ///     let intervalState =
signposter.beginInterval("Example", "An
unsigned integer \(x,
privacy: .private)")
    ///
    /// - Warning: Do not explicity
create SignpostMetadata. Instead pass a
string interpolation.
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    ///   - message: A string
interpolation that represents the message
you want to add to the signposts.
    ///
    /// - Returns: an open
`OSSignpostIntervalState` for the
interval. Pass this to the corresponding
end interval call.
    public func beginInterval(_ name:
StaticString, id: OSSignpostID
= .exclusive, _ message:
SignpostMetadata) ->
OSSignpostIntervalState

    /// Begins a signposted interval and
returns an open
`OSSignpostIntervalState`.
```

```
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///     let intervalState =
signposter.beginInterval("Example", id:
signpostID)
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    ///
    /// - Returns: an open
`OSSignpostIntervalState` for the
interval. Pass this to the corresponding
end interval call.
    public func beginInterval(_ name:
StaticString, id: OSSignpostID
= .exclusive) -> OSSignpostIntervalState

    /// Begins a signposted animation
interval and returns an open
`OSSignpostIntervalState`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///     let intervalState =
signposter.beginAnimationInterval("Exampl
e", id: signpostID)
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    ///
    /// - Returns: an open
`OSSignpostIntervalState` for the
interval. Pass this to the corresponding
end interval call.
    public func beginAnimationInterval(_
name: StaticString, id: OSSignpostID
= .exclusive) -> OSSignpostIntervalState

    /// Begins a signposted animation
interval and returns an open
`OSSignpostIntervalState`.
    ///
    /// Values that can be interpolated
```

```
in the message parameter include signed
and unsigned Swift integers,
    /// Floats, Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///     let intervalState =
signposter.beginAnimationInterval("Exampl
e", id: signpostID, "A string
interpolation \(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     let intervalState =
signposter.beginAnimationInterval("Exampl
e", "An unsigned integer \(x,
format: .hex, align: .right(columns:
10))")
    ///     let intervalState =
signposter.beginAnimationInterval("Exampl
e", "An unsigned integer \(x,
privacy: .private)")
    ///
    /// - Warning: Do not explicity
create SignpostMetadata. Instead pass a
string interpolation.
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    ///   - message: A string
interpolation that represents the message
you want to add to the signposts.
    ///
    /// - Returns: an open
`OSSignpostIntervalState` for the
interval. Pass this to the corresponding
end interval call.
```

```
    public func beginAnimationInterval(_
name: StaticString, id: OSSignpostID
= .exclusive, _ message:
SignpostMetadata) ->
OSSignpostIntervalState

    /// Ends the signposted interval
corresponding to the consumed
`OSSignpostIntervalState`.
    ///
    /// Values that can be interpolated
in the message parameter include signed
and unsigned Swift integers,
    /// Floats, Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///     let intervalState =
signposter.beginInterval("Example", id:
signpostID)
    ///     // ...
    ///
```

```
signposter.endInterval("Example",
intervalState, "A string interpolation \
(x)")
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///
signposter.endInterval("Example",
intervalState, "An unsigned integer \(x,
format: .hex, align: .right(columns:
10))")
    ///
signposter.endInterval("Example",
intervalState, "An unsigned integer \(x,
privacy: .private)")
    ///
    /// - Warning: Do not explicity
create SignpostMetadata. Instead pass a
string interpolation.
    ///
    /// - Parameters:
```

```
    ///   - name: The name of this event.
    ///   - state: The consumed
`OSSignpostIntervalState` produced by
`beginInterval`.
    ///   - message: A string
interpolation that represents the message
you want to add to the signposts.
    public func endInterval(_ name:
StaticString, _ state:
OSSignpostIntervalState, _ message:
SignpostMetadata)

    /// Ends the signposted interval
corresponding to the consumed
`OSSignpostIntervalState`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///     let intervalState =
signposter.beginInterval("Example", id:
signpostID)
    ///     // ...
    ///
signposter.endInterval("Example",
intervalState)
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - state: The consumed
`OSSignpostIntervalState` produced by
`beginInterval`.
    public func endInterval(_ name:
StaticString, _ state:
OSSignpostIntervalState)

    /// Begins and ends a signposted
interval around the execution of a
closure.
    ///
    /// Values that can be interpolated
in the message parameter include signed
and unsigned Swift integers,
    /// Floats, Doubles, Bools, Strings,
NSObjects, UnsafeRaw(Buffer)Pointers,
values conforming to
    /// `CustomStringConvertible` like
Arrays and Dictionaries, and metatypes
like
    /// `type(of: c)`, `Int.self`.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///
signposter.withIntervalSignpost("Example"
, id: signpostID) {
    ///       // perform a task
    ///     }
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
=========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///
signposter.withIntervalSignpost("Example"
, "An unsigned integer \(x,
privacy: .private)") {
    ///       /* do the task */
    ///     }
    ///
    /// - Warning: Do not explicity
create SignpostMetadata. Instead pass a
string interpolation.
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
```

```
for this parameter, this method does
nothing.
    ///   - message: A string
interpolation that represents the message
you want to add to the signposts.
    ///   - around: A closure around
which an interval is signposted.
    public func withIntervalSignpost<T>(_
name: StaticString, id: OSSignpostID
= .exclusive, _ message:
SignpostMetadata, around task: () throws
-> T) rethrows -> T

    /// Begins and ends a signposted
interval around the execution of a
closure.
    ///
    /// Examples
    /// ========
    ///
    ///     let signposter =
OSSignposter()
    ///     let signpostID =
signposter.makeSignpostID()
    ///
signposter.withIntervalSignpost("Example"
, id: signpostID) {
    ///       // perform a task
    ///     }
    ///
    /// Formatting Interpolated
Expressions and Specifying Privacy
    ///
```

```
==========================================
=================
    ///
    /// Formatting and privacy options
for the interpolated values can be passed
as arguments
    /// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
    /// default values.
    ///
    ///     let signpostID =
signposter.makeSignpostID()
    ///
signposter.withIntervalSignpost("Example"
, id: signpostID) {
    ///       /* do the task */
    ///     }
    ///
    /// - Parameters:
    ///   - name: The name of this event.
    ///   - id: A signpost identifier you
use to disambiguate between signposts
with the same name.
    ///   If you specify invalid or null
for this parameter, this method does
nothing.
    ///   - around: A closure around
which an interval is signposted.
    public func withIntervalSignpost<T>(_
name: StaticString, id: OSSignpostID
= .exclusive, around task: () throws ->
T) rethrows -> T
```

```
    /// Returns an OSSignpostID that is
unique among those generated by this
`OSSignposter`.
    @inlinable public func
makeSignpostID() -> OSSignpostID

    /// Generates an `OSSignpostID` from
an object.
    ///
    /// - Parameter from: any object that
disambiguates among intervals made with
the same
    /// `OSSignposter` and interval names
    ///
    /// - Returns: an `OSSignpostID` that
uniquely corresponds to the object.
    @inlinable public func
makeSignpostID(from object: AnyObject) ->
OSSignpostID
}

@available(macOS 12.0, iOS 15.0, watchOS
8.0, tvOS 15.0, *)
public typealias SignpostMetadata =
OSLogMessage

/// Maximum number of arguments i.e.,
interpolated expressions that can
/// be used in the string interpolations
passed to the log APIs.
/// This limit is imposed by the logging
system.
```

```swift
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@inlinable public var
maxOSLogArgumentCount: UInt8 { get }

@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
public func os_log(_ type: OSLogType,
dso: UnsafeRawPointer = #dsohandle, log:
OSLog = .default, _ message:
StaticString, _ args: any CVarArg...)

@available(macOS 10.12, iOS 10.0, watchOS
3.0, tvOS 10.0, *)
public func os_log(_ message:
StaticString, dso: UnsafeRawPointer? =
#dsohandle, log: OSLog = .default, type:
OSLogType = .default, _ args: any
CVarArg...)

/// Logs a string interpolation to the
default subsystem at the default level.
///
/// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
/// Doubles, Bools, Strings, NSObjects,
UnsafeRaw(Buffer)Pointers, values
conforming to
/// `CustomStringConvertible` like Arrays
and Dictionaries, and metatypes like
/// `type(of: c)`, `Int.self`.
///
```

```
/// Examples
/// ========
///
///     os_log("A string interpolation \
(x)")
///
/// Formatting Interpolated Expressions
and Specifying Privacy
///
=========================================
=================
///
/// Formatting and privacy options for
the interpolated values can be passed as
arguments
/// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
/// default values.
///
///     os_log("An unsigned integer \(x,
format: .hex, align: .right(columns:
10))")
///     os_log("An unsigned integer \(x,
privacy: .private)")
///
/// - Warning: Do not explicity create
OSLogMessage. Instead pass a string
interpolation.
///
/// - Parameter message: A string
interpolation.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public func os_log(_ message:
OSLogMessage)

/// Logs a string interpolation to the
logging system, optionally specifying a
custom
/// log object and a log level.
///
/// Values that can be interpolated
include signed and unsigned Swift
integers, Floats,
/// Doubles, Bools, Strings, NSObjects,
UnsafeRaw(Buffer)Pointers, values
conforming to
/// `CustomStringConvertible` like Arrays
and Dictionaries, and metatypes like
/// `type(of: c)`, `Int.self`.
///
/// Examples
/// ========
///
///     os_log(.info, "A string
interpolation \(x)")
///     os_log(.debug, log: customLog, "A
string interpolation \(x)")
///
/// Formatting Interpolated Expressions
and Specifying Privacy
///
=========================================
=================
///
```

```
/// Formatting and privacy options for
the interpolated values can be passed as
arguments
/// to the interpolations. These are
optional arguments. When not specified,
they will be set to their
/// default values.
///
///     os_log(
///       .error,
///       log: customLog,
///       "An unsigned integer \(x,
format: .hex, align: .right(columns:
10))")
///
///     os_log(.fault, log: customLog,
"unsigned value \(x, privacy: .private)")
///
/// - Warning: Do not explicity create
OSLogMessage. Instead pass a string
interpolation.
///
/// - Parameters:
///   - logLevel: Logging level.
///   - logObject: An instance of
`OSLog`.
///   - message: A string interpolation.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public func os_log(_ logLevel: OSLogType
= .default, log logObject: OSLog
= .default, _ message: OSLogMessage)
```

```
@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
public func os_signpost(_ type:
OSSignpostType, dso: UnsafeRawPointer =
#dsohandle, log: OSLog, name:
StaticString, signpostID: OSSignpostID
= .exclusive)

@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
public func os_signpost(_ type:
OSSignpostType, dso: UnsafeRawPointer =
#dsohandle, log: OSLog, name:
StaticString, signpostID: OSSignpostID
= .exclusive, _ format: StaticString, _
arguments: any CVarArg...)

/// Begin an os_signpost, tagged as
animation, with a message.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
public func os_signpost(_ animationBegin:
OSSignpostAnimationBegin, dso:
UnsafeRawPointer = #dsohandle, log:
OSLog, name: StaticString, signpostID:
OSSignpostID = .exclusive, _ format:
AnimationFormatString.OSLogMessage, _
arguments: any CVarArg...)

/// Begin an os_signpost, tagged as
animation, without a message.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
```

```
public func os_signpost(_ animationBegin:
OSSignpostAnimationBegin, dso:
UnsafeRawPointer = #dsohandle, log:
OSLog, name: StaticString, signpostID:
OSSignpostID = .exclusive)

@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
extension OSSignpostType {

    public static let event:
OSSignpostType

    public static let begin:
OSSignpostType

    public static let end: OSSignpostType
}

@available(macOS 10.14, iOS 12.0, watchOS
5.0, tvOS 12.0, *)
extension OSLog {

    public struct Category : Sendable {

        public let rawValue: String

        public static let
pointsOfInterest: OSLog.Category
    }

    public convenience init(subsystem:
String, category: OSLog.Category)
```

```
    public var signpostsEnabled: Bool {
get }
}

extension OSLogType {

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let `default`:
OSLogType

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let info: OSLogType

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let debug: OSLogType

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let error: OSLogType

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let fault: OSLogType
}

@available(macOS 10.12, iOS 10.0, watchOS
3.0, tvOS 10.0, *)
extension OSLog : @unchecked Sendable {
    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let disabled: OSLog

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public static let `default`: OSLog

    @available(macOS 10.12, iOS 10.0,
watchOS 3.0, tvOS 10.0, *)
    public convenience init(subsystem:
String, category: String)
}

@available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
extension WorkGroup {

    @available(macOS 11.0, *)
    public func copyPort() -> mach_port_t

    @available(macOS 11.0, *)
    public convenience init?(port:
mach_port_t, name: String? = nil)

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func copy(name: String? = nil)
-> WorkGroup?

    public struct JoinToken {
    }
    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func join() ->
WorkGroup.JoinToken

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func leave(token:
WorkGroup.JoinToken)

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func cancel()

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public var isCancelled: Bool { get }

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public var maxParallelThreads: Int {
get }

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func setWorkingArena(arena:
UnsafeMutableRawPointer?, max_workers:
UInt32, destruct: @convention(c)
(UnsafeMutableRawPointer?) -> Void)

    public typealias Index = UInt32

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public var workingArena:
(UnsafeMutableRawPointer?,
WorkGroup.Index) { get }
}

extension WorkGroup : Repeatable {

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func start(at timestamp:
UInt64, deadline: UInt64)

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func updateDeadline(deadline:
UInt64)

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public func finish()
}

extension WorkGroupParallel {

    @available(macOS 11.0, iOS 14.0, tvOS
14.0, watchOS 7.0, *)
    public convenience init?(name:
String? = nil)
}

/// Maximum number of arguments i.e.,
interpolated expressions that can
```

```
/// be used in the string interpolations
passed to the log APIs.
/// This limit is imposed by the logging
system.
@available(macOS 11.0, iOS 14.0, watchOS
7.0, tvOS 14.0, *)
@inlinable public let
maxOSLogArgumentCount: UInt8 { get }
```