```
/// A type that emits notifications to
observers when underlying data changes.
///
/// Conforming to this protocol signals
to other APIs that the type supports
/// observation. However, applying the
`Observable` protocol by itself to a
/// type doesn't add observation
functionality to the type. Instead,
always use
/// the ``Observation/Observable()``
macro when adding observation
/// support to a type.
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
public protocol Observable {
}

/// Defines and implements conformance of
the Observable protocol.
///
/// This macro adds observation support
to a custom type and conforms the type
/// to the ``Observation/Observable``
protocol. For example, the following code
/// applies the `Observable` macro to the
type `Car` making it observable:
///
///     @Observable
///     class Car {
///        var name: String = ""
///        var needsRepairs: Bool = false
```

```
///
///         init(name: String,
needsRepairs: Bool = false) {
///             self.name = name
///             self.needsRepairs =
needsRepairs
///         }
///     }
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
@attached(member, names:
named(_$observationRegistrar),
named(access), named(withMutation))
@attached(memberAttribute)
@attached(extension, conformances:
Observable) public macro Observable() =
#externalMacro(module:
"ObservationMacros", type:
"ObservableMacro")

/// Disables observation tracking of a
property.
///
/// By default, an object can observe any
property of an observable type that
/// is accessible to the observing
object. To prevent observation of an
/// accessible property, attach the
`ObservationIgnored` macro to the
property.
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
@attached(accessor, names:
```

```swift
named(willSet)) public macro
ObservationIgnored() =
#externalMacro(module:
"ObservationMacros", type:
"ObservationIgnoredMacro")

/// Provides storage for tracking and
access to data changes.
///
/// You don't need to create an instance
of `ObservationRegistrar` when using
/// the ``Observation/Observable()``
macro to indicate observability of a
type.
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
public struct ObservationRegistrar :
Sendable {

    /// Creates an instance of the
observation registrar.
    ///
    /// You don't need to create an
instance of
    ///
``Observation/ObservationRegistrar`` when
using the
    /// ``Observation/Observable()``
macro to indicate observably
    /// of a type.
    public init()

    /// Registers access to a specific
```

```
property for observation.
    ///
    /// - Parameters:
    ///   - subject: An instance of an
observable type.
    ///   - keyPath: The key path of an
observed property.
    public func access<Subject, Member>(_
subject: Subject, keyPath:
KeyPath<Subject, Member>) where Subject :
Observable

    /// A property observation called
before setting the value of the subject.
    ///
    /// - Parameters:
    ///     - subject: An instance of an
observable type.
    ///     - keyPath: The key path of an
observed property.
    public func willSet<Subject,
Member>(_ subject: Subject, keyPath:
KeyPath<Subject, Member>) where Subject :
Observable

    /// A property observation called
after setting the value of the subject.
    ///
    /// - Parameters:
    ///   - subject: An instance of an
observable type.
    ///   - keyPath: The key path of an
observed property.
```

```
    public func didSet<Subject, Member>(_
subject: Subject, keyPath:
KeyPath<Subject, Member>) where Subject :
Observable

    /// Identifies mutations to the
transactions registered for observers.
    ///
    /// This method calls
``willset(_:keypath:)`` before the
mutation. Then it
    /// calls ``didset(_:keypath:)``
after the mutation.
    /// - Parameters:
    ///   - of: An instance of an
observable type.
    ///   - keyPath: The key path of an
observed property.
    public func withMutation<Subject,
Member, T>(of subject: Subject, keyPath:
KeyPath<Subject, Member>, _ mutation: ()
throws -> T) rethrows -> T where
Subject : Observable
}

@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
extension ObservationRegistrar : Codable
{

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

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

@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
extension ObservationRegistrar : Hashable
{

    /// Returns a Boolean value
```

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

/// Synthesizes a property for accessors.
///
/// The ``Observation`` module uses this
macro. Its use outside of the
/// framework isn't necessary.
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
@attached(accessor, names: named(init),
named(get), named(set), named(_modify))
@attached(peer, names: prefixed(`_`))
public macro ObservationTracked() =
#externalMacro(module:
"ObservationMacros", type:
"ObservationTrackedMacro")

/// Tracks access to properties.
///
/// This method tracks access to any
property within the `apply` closure, and
/// informs the caller of value changes
made to participating properties by way
/// of the `onChange` closure. For
example, the following code tracks
changes
/// to the name of cars, but it doesn't
track changes to any other property of
/// `Car`:
///
///     func render() {
///         withObservationTracking {
///             for car in cars {
///                 print(car.name)
///             }
///         } onChange: {
///             print("Schedule
renderer.")
///         }
///     }
///
```

```
/// - Parameters:
///     - apply: A closure that contains
properties to track.
///     - onChange: The closure invoked
when the value of a property changes.
///
/// - Returns: The value that the `apply`
closure returns if it has a return
/// value; otherwise, there is no return
value.
@available(macOS 14.0, iOS 17.0, watchOS
10.0, tvOS 17.0, *)
public func withObservationTracking<T>(_
apply: () -> T, onChange: @autoclosure ()
-> @Sendable () -> Void) -> T
```