```
import DeveloperToolsSupport
import Foundation
import Photos
import
PhotosUI.PHContentEditingController
import PhotosUI.PHLivePhotoView
import PhotosUI.PHPicker
import PhotosUI.PHProjectExtensionContext
import
PhotosUI.PHProjectExtensionController
import PhotosUI.PHProjectInfo
import PhotosUI.PHProjectTypeDescription
import
PhotosUI.PHProjectTypeDescriptionDataSour
ce
import PhotosUI.PhotosUITypes
import TargetConditionals
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

/// A configuration for
`PHPickerViewController`.
@available(iOS 14.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
public struct PHPickerConfiguration :
Equatable, Hashable, @unchecked Sendable
{

    /// A mode that determines which
representation `PHPickerViewController`
should provide for an asset given a type
```

```
identifier, if multiple representations
are available.
    public enum AssetRepresentationMode :
Sendable {

        /// Uses the best representation
determined by the system. This may change
in future releases.
        case automatic

        /// Uses the current
representation to avoid transcoding if
possible.
        case current

        /// Uses the most compatible
representation if possible, even if
transcoding is required.
        case compatible

        /// 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:
```

```
PHPickerConfiguration.AssetRepresentation
Mode, b:
PHPickerConfiguration.AssetRepresentation
Mode) -> 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 }
    }

    /// An enum that determines how
`PHPickerViewController` handles user
selection.
    @available(iOS 15.0, *)
    public enum Selection : Sendable {

        /// Uses the default selection
behavior.
        case `default`
```

```
        /// Uses the selection order made
by the user. Selected assets are
numbered.
        case ordered

        /// Selection can be delivered
continuously.
        @available(iOS 17.0, macOS 14.0,
*)
        @available(watchOS, unavailable)
        case continuous

        /// Selection can be delivered
continuously and uses the selection order
made by the user. Selected assets are
numbered.
        @available(iOS 17.0, macOS 14.0,
*)
        @available(watchOS, unavailable)
        case continuousAndOrdered

        /// 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:
PHPickerConfiguration.Selection, b:
PHPickerConfiguration.Selection) -> 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 }
    }

    /// An update configuration for
`PHPickerViewController`.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    public struct Update : Equatable,
Hashable, Sendable {

        /// The maximum number of assets
that can be selected. Default is `nil`.
        public var selectionLimit: Int?
```

```
        /// Edges of the picker that have
no margin between the content and the
edge (e.g. without bars in between).
Default is `nil`.
        public var
edgesWithoutContentMargins:
NSDirectionalRectEdge?

        public init()

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

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

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

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

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

    /// The preferred representation mode
of selected assets. Default is
`.automatic`.
    ///
    /// Setting
`preferredAssetRepresentationMode` to
`.automatic` means the best
representation determined by the system
will be used.
    public var
preferredAssetRepresentationMode:
PHPickerConfiguration.AssetRepresentation
Mode

    /// The selection behavior of the
picker. Default is `.default`.
    @available(iOS 15.0, *)
    public var selection:
PHPickerConfiguration.Selection

    /// The maximum number of assets that
can be selected. Default is 1.
```

```
    ///
    /// Setting `selectionLimit` to 0
means maximum supported by the system.
    public var selectionLimit: Int

    /// Types of assets that can be
shown. Default is `nil`.
    ///
    /// Setting `filter` to `nil` means
all asset types can be shown.
    public var filter: PHPickerFilter?

    /// Local identifiers of assets to be
shown as selected when the picker is
presented. Default is an empty array.
    ///
    /// `preselectedAssetIdentifiers`
should be an empty array if
`selectionLimit` is 1 or `photoLibrary`
is not specified. Returned item providers
for preselected assets are always empty.
    @available(iOS 15.0, *)
    public var
preselectedAssetIdentifiers: [String]

    /// The mode of the picker. Default
is `.default`.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    public var mode: PHPickerMode

    /// Edges of the picker that have no
margin between the content and the edge
```

```
(e.g. without bars in between). Default
is `[]`.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    public var
edgesWithoutContentMargins:
NSDirectionalRectEdge

    /// Capabilities of the picker that
should be disabled. Default is `[]`.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    public var disabledCapabilities:
PHPickerCapabilities

    /// Initializes a new configuration
with the system photo library. This
configuration never returns asset
identifiers.
    public init()

    /// Initializes a new configuration
with the `photoLibrary` the picker should
use.
    public init(photoLibrary:
PHPhotoLibrary)

    /// 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:
PHPickerConfiguration, b:
PHPickerConfiguration) -> 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(iOS 14.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
extension
PHPickerConfiguration.AssetRepresentation
Mode : Equatable {
}

@available(iOS 14.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
```

```
extension
PHPickerConfiguration.AssetRepresentation
Mode : Hashable {
}

@available(macOS 13.0, iOS 15.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
extension PHPickerConfiguration.Selection
: Equatable {
}

@available(macOS 13.0, iOS 15.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
extension PHPickerConfiguration.Selection
: Hashable {
}

/// A filter that restricts which types
of assets `PHPickerViewController` can
show.
@available(iOS 14.0, macOS 13.0, watchOS
9.0, *)
@available(tvOS, unavailable)
public struct PHPickerFilter : Equatable,
Hashable, @unchecked Sendable {

    /// The filter for images.
    public static let images:
PHPickerFilter

    /// The filter for videos.
```

```
    @available(watchOS, unavailable)
    public static let videos:
PHPickerFilter

    /// The filter for live photos.
    @available(watchOS, unavailable)
    public static let livePhotos:
PHPickerFilter

    /// The filter for Depth Effect
photos.
    @available(iOS 16.0, *)
    @available(watchOS, unavailable)
    public static let depthEffectPhotos:
PHPickerFilter

    /// The filter for bursts.
    @available(iOS 16.0, *)
    @available(watchOS, unavailable)
    public static let bursts:
PHPickerFilter

    /// The filter for panorama photos.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static let panoramas:
PHPickerFilter

    /// The filter for screenshots.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static let screenshots:
PHPickerFilter
```

```
    /// The filter for screen recordings.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static let screenRecordings:
PHPickerFilter

    /// The filter for Slow-Mo videos.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static let slomoVideos:
PHPickerFilter

    /// The filter for time-lapse videos.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static let timelapseVideos:
PHPickerFilter

    /// The filter for Cinematic videos.
    @available(iOS 16.0, *)
    @available(watchOS, unavailable)
    public static let cinematicVideos:
PHPickerFilter

    /// The filter for spatial media.
    @available(iOS 18.0, macOS 15.0,
visionOS 2.0, *)
    @available(watchOS, unavailable)
    public static let spatialMedia:
PHPickerFilter

    /// Returns a new filter based on the
```

```
asset playback style.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static func playbackStyle(_
playbackStyle: PHAsset.PlaybackStyle) ->
PHPickerFilter

    /// Returns a new filter formed by
OR-ing the filters in a given array.
    @available(watchOS, unavailable)
    public static func any(of subfilters:
[PHPickerFilter]) -> PHPickerFilter

    /// Returns a new filter formed by
AND-ing the filters in a given array.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static func all(of subfilters:
[PHPickerFilter]) -> PHPickerFilter

    /// Returns a new filter formed by
negating the given filter.
    @available(iOS 15.0, *)
    @available(watchOS, unavailable)
    public static func not(_ filter:
PHPickerFilter) -> PHPickerFilter

    /// 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:
PHPickerFilter, b: PHPickerFilter) ->
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 }
}

/// An enum that determines the mode of
`PHPickerViewController`.
@available(iOS 17.0, macOS 14.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
public struct PHPickerMode : Equatable,
Hashable, Sendable {

    /// Default picker mode.
    public static let `default`:
PHPickerMode
```

```
    /// Compact picker mode (single row).
    @available(watchOS, unavailable)
    public static var compact:
PHPickerMode

    /// 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:
PHPickerMode, b: PHPickerMode) -> 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 }
}
```

```
/// A user selected asset from
`PHPickerViewController`.
@available(iOS 14.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
public struct PHPickerResult : Equatable,
Hashable {

    /// Representations of the selected
asset.
    public let itemProvider:
NSItemProvider

    /// The local identifier of the
selected asset.
    public var assetIdentifier: String? {
get }

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

/// A set of methods that the delegate
must implement to respond to
`PHPickerViewController` user events.
@available(iOS 14.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
@MainActor @preconcurrency public
protocol PHPickerViewControllerDelegate :
AnyObject {

    /// Called when the user completes a
selection or dismisses
`PHPickerViewController` using the cancel
button.
    ///
    /// The picker won't be automatically
dismissed when this method is called.
    @MainActor @preconcurrency func
picker(_ picker: PHPickerViewController,
didFinishPicking results:
```

```
[PHPickerResult])
}

@available(iOS 14.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
extension PHPickerViewController {

    /// The configuration passed in
during initialization.
    @MainActor @preconcurrency public var
configuration: PHPickerConfiguration {
get }

    /// The delegate to be notified.
    @MainActor @preconcurrency weak
public var delegate: (any
PHPickerViewControllerDelegate)?

    /// Initializes a new picker with the
`configuration` the picker should use.
    @MainActor @preconcurrency public
convenience init(configuration:
PHPickerConfiguration)

    /// Updates the picker using the
configuration.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    @MainActor @preconcurrency public
func updatePicker(using configuration:
PHPickerConfiguration.Update)
}
```

```
// MARK: - SwiftUI Additions

import CoreTransferable
import SwiftUI
import UniformTypeIdentifiers

// Available when SwiftUI is imported
with PhotosUI
/// A control that allows a user to
choose photos and/or videos from the
photo library.
///
/// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
@available(iOS 16.0, macOS 13.0, watchOS
9.0, *)
@available(tvOS, unavailable)
@MainActor @preconcurrency public struct
PhotosPicker<Label> : View where Label :
View {

    /// Creates a Photos picker that
selects a `PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
```

```
    /// - Parameters:
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - label: The view that
describes the action of choosing an item
from the photo library.
    @preconcurrency nonisolated public
init(selection: Binding<PhotosPickerItem?
>, matching filter: PHPickerFilter? =
nil, preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, @ViewBuilder label:
@Sendable () -> Label)

    /// Creates a Photos picker that
selects a collection of
`PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
```

```
    /// - Parameters:
    ///     - selection: All items being
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - label: The view that
describes the action of choosing items
from the photo library.
    @preconcurrency nonisolated public
init(selection:
Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
```

```
licy = .automatic, @ViewBuilder label:
@Sendable () -> Label)

    /// Creates a Photos picker that
selects a `PhotosPickerItem` from a given
photo library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    ///     - label: The view that
describes the action of choosing an item
from the photo library.
    @available(watchOS, unavailable)
    @preconcurrency nonisolated public
init(selection: Binding<PhotosPickerItem?
```

```
>, matching filter: PHPickerFilter? =
nil, preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary, @ViewBuilder label:
@Sendable () -> Label)

    /// Creates a Photos picker that
selects a collection of
`PhotosPickerItem` from a given photo
library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - selection: All items being
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
```

```
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    ///     - label: The view that
describes the action of choosing items
from the photo library.
    @available(watchOS, unavailable)
    @preconcurrency nonisolated public
init(selection:
Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary, @ViewBuilder label:
@Sendable () -> Label)

    /// The content and behavior of the
view.
    ///
    /// When you implement a custom view,
you must implement a computed
    /// `body` property to provide the
content for your view. Return a view
    /// that's composed of built-in views
that SwiftUI provides, plus other
```

```
    /// composite views that you've
already defined:
    ///
    ///     struct MyView: View {
    ///         var body: some View {
    ///             Text("Hello, World!")
    ///         }
    ///     }
    ///
    /// For more information about
composing views and a view hierarchy,
    /// see <doc:Declaring-a-Custom-
View>.
    @MainActor @preconcurrency public var
body: some View { get }

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 16.0, watchOS 9.0,
macOS 13.0, *)
    @available(tvOS, unavailable)
    public typealias Body = some View
}

// Available when SwiftUI is imported
with PhotosUI
@available(iOS 16.0, macOS 13.0, watchOS
9.0, *)
```

```
@available(tvOS, unavailable)
extension PhotosPicker where Label ==
Text {

    /// Creates a Photos picker with its
label generated from a localized string
key that selects a `PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - titleKey: A localized
string key that describes the purpose of
showing the picker.
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    nonisolated public init(_ titleKey:
LocalizedStringKey, selection:
Binding<PhotosPickerItem?>, matching
filter: PHPickerFilter? = nil,
```

```
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic)

    /// Creates a Photos picker with its
label generated from a string that
selects a `PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - title: A string that
describes the purpose of showing the
picker.
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    nonisolated public init<S>(_ title:
S, selection: Binding<PhotosPickerItem?>,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
```

```
PhotosPickerItem.EncodingDisambiguationPolicy = .automatic) where S : StringProtocol

    /// Creates a Photos picker with its label generated from a localized string key that selects a collection of `PhotosPickerItem`.
    ///
    /// The user explicitly grants access only to items they choose, so photo library access authorization is not needed.
    ///
    /// - Parameters:
    ///     - titleKey: A localized string key that describes the purpose of showing the picker.
    ///     - selection: All items being shown and selected in the Photos picker.
    ///     - maxSelectionCount: The maximum number of items that can be selected. Default is `nil`. Setting it to `nil` means maximum supported by the system.
    ///     - selectionBehavior: The selection behavior of the Photos picker. Default is `.default`.
    ///     - filter: Types of items that can be shown. Default is `nil`. Setting it to `nil` means all supported types can be shown.
    ///     - preferredItemEncoding: The
```

```
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    nonisolated public init(_ titleKey:
LocalizedStringKey, selection:
Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic)

    /// Creates a Photos picker with its
label generated from a string that
selects a collection of
`PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - title: A string that
describes the purpose of showing the
picker.
    ///     - selection: All items being
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
```

```
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    nonisolated public init<S>(_ title:
S, selection:
Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic) where S :
StringProtocol

    /// Creates a Photos picker with its
label generated from a localized string
key that selects a `PhotosPickerItem`
from a given photo library.
```

```
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - titleKey: A localized
string key that describes the purpose of
showing the picker.
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    @available(watchOS, unavailable)
    nonisolated public init(_ titleKey:
LocalizedStringKey, selection:
Binding<PhotosPickerItem?>, matching
filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary)
```

```
    /// Creates a Photos picker with its
label generated from a string that
selects a `PhotosPickerItem` from a given
photo library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - title: A string that
describes the purpose of showing the
picker.
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    @available(watchOS, unavailable)
    nonisolated public init<S>(_ title:
S, selection: Binding<PhotosPickerItem?>,
matching filter: PHPickerFilter? = nil,
```

```
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary) where S : StringProtocol

    /// Creates a Photos picker with its
label generated from a localized string
key that selects a collection of
`PhotosPickerItem` from a given photo
library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - titleKey: A localized
string key that describes the purpose of
showing the picker.
    ///     - selection: All items being
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
```

```
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    @available(watchOS, unavailable)
    nonisolated public init(_ titleKey:
LocalizedStringKey, selection:
Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary)

    /// Creates a Photos picker with its
label generated from a string that
selects a collection of
`PhotosPickerItem` from a given photo
library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
```

```
    /// - Parameters:
    ///     - title: A string that
describes the purpose of showing the
picker.
    ///     - selection: All items being
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    @available(watchOS, unavailable)
    nonisolated public init<S>(_ title:
S, selection:
Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
```

```
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary) where S : StringProtocol
}

// Available when SwiftUI is imported
with PhotosUI
@available(iOS 16.0, macOS 13.0, watchOS
9.0, *)
@available(tvOS, unavailable)
extension PhotosPicker : Sendable {
}

// Available when SwiftUI is imported
with PhotosUI
/// An item that can be provided to or
provided by the Photos picker.
///
/// An item can contain multiple
representations. Each representation has
a corresponding content type.
@available(iOS 16.0, macOS 13.0, watchOS
9.0, *)
@available(tvOS, unavailable)
public struct PhotosPickerItem :
Equatable, Hashable, @unchecked Sendable
{

    /// A policy that decides the
encoding to use given a content type, if
multiple encodings are available.
```

```
    public struct
EncodingDisambiguationPolicy : Equatable,
Hashable, Sendable {

        /// Uses the best encoding
determined by the system. This may change
in future releases.
        public static let automatic:
PhotosPickerItem.EncodingDisambiguationPo
licy

        /// Uses the current encoding to
avoid transcoding if possible.
        public static let current:
PhotosPickerItem.EncodingDisambiguationPo
licy

        /// Uses the most compatible
encoding if possible, even if transcoding
is required.
        public static let compatible:
PhotosPickerItem.EncodingDisambiguationPo
licy

        /// 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:
PhotosPickerItem.EncodingDisambiguationPo
licy, b:
PhotosPickerItem.EncodingDisambiguationPo
licy) -> 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 }
    }

    /// The local identifier of the item.
It will be `nil` if the Photos picker is
created without a photo library.
    @available(watchOS, unavailable)
    public var itemIdentifier: String? {
get }
```

```
    /// All supported content types of
the item, in order of most preferred to
least preferred.
    public var supportedContentTypes:
[UTType] { get }

    /// Loads a `Transferable` object
using a representation of the item by
matching content types.
    ///
    /// The representation corresponding
to the first matching content type of the
item will be used.
    /// If multiple encodings are
available for the matched content type,
the preferred item encoding provided to
the Photos picker decides which encoding
to use.
    /// An error will be returned if the
`Transferable` object doesn't support any
of the supported content types of the
item.
    ///
    /// - Parameters:
    ///     - type: The actual type of
the `Transferable` object.
    ///     - completionHandler: The
closure to be called when the
`Transferable` object is loaded (`nil` if
no supported content type is found) or an
error is encountered.
    @discardableResult
```

```
    @preconcurrency public func
loadTransferable<T>(type: T.Type,
completionHandler: @escaping @Sendable
(Result<T?, any Error>) -> Void) ->
Progress where T : Transferable

    /// Loads a `Transferable` object
using a representation of the item by
matching content types.
    ///
    /// The representation corresponding
to the first matching content type of the
item will be used.
    /// If multiple encodings are
available for the matched content type,
the preferred item encoding provided to
the Photos picker decides which encoding
to use.
    /// An error will be thrown if the
`Transferable` object doesn't support any
of the supported content types of the
item.
    ///
    /// - Parameters:
    ///     - type: The actual type of
the `Transferable` object.
    /// - Throws: The encountered error
while loading the `Transferable` object.
    /// - Returns: The loaded
`Transferable` object, or `nil` if no
supported content type is found.
    public func loadTransferable<T>(type:
T.Type) async throws -> sending T? where
```

```
T : Transferable

    /// Creates an item without any
representation using an identifier.
    ///
    /// - Parameters:
    ///     - itemIdentifier: The local
identifier of the item.
    @available(watchOS, unavailable)
    public init(itemIdentifier: String)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (a:
PhotosPickerItem, b: PhotosPickerItem) ->
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 when SwiftUI is imported
with PhotosUI
/// A value that determines how the
Photos picker handles user selection.
@available(iOS 16.0, macOS 13.0, watchOS
9.0, *)
@available(tvOS, unavailable)
public struct
PhotosPickerSelectionBehavior :
Equatable, Hashable, Sendable {

    /// Uses the default selection
behavior.
    public static let `default`:
PhotosPickerSelectionBehavior

    /// Uses the selection order made by
the user. Selected items are numbered.
    public static let ordered:
PhotosPickerSelectionBehavior

    /// Selection can be delivered
continuously.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    public static let continuous:
PhotosPickerSelectionBehavior
```

```
    /// Selection can be delivered
continuously and uses the selection order
made by the user. Selected assets are
numbered.
    @available(iOS 17.0, macOS 14.0, *)
    @available(watchOS, unavailable)
    public static let
continuousAndOrdered:
PhotosPickerSelectionBehavior

    /// 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:
PhotosPickerSelectionBehavior, b:
PhotosPickerSelectionBehavior) -> 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 when SwiftUI is imported
with PhotosUI
/// A value that determines the style of
the Photos picker.
@available(iOS 17.0, macOS 14.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
public struct PhotosPickerStyle :
Equatable, Hashable, Sendable {

    /// Modal picker mode.
    public static let presentation:
PhotosPickerStyle

    /// Inline picker mode.
    public static let inline:
PhotosPickerStyle

    /// Inline and compact picker mode
(single row).
    public static let compact:
PhotosPickerStyle

    /// 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:
PhotosPickerStyle, b: PhotosPickerStyle)
-> 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 when SwiftUI is imported
with PhotosUI
@available(iOS 17.0, macOS 14.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
extension View {

    /// Sets the mode of the Photos
picker.
```

```
    ///
    /// - Parameters:
    ///     - mode: One of the available
modes.
    /// - Returns: A Photos picker that
uses the specified mode.
    nonisolated public func
photosPickerStyle(_ style:
PhotosPickerStyle) -> some View


    /// Sets the accessory visibility of
the Photos picker. Accessories include
anything between the content and the
edge, like the navigation bar or the
sidebar.
    ///
    /// - Parameters:
    ///     - edges: The accessory
visibility to apply.
    ///     - edges: One or more of the
available edges.
    /// - Returns: A Photos picker with
the specified accessory visibility.
    nonisolated public func
photosPickerAccessoryVisibility(_
visibility: Visibility, edges: Edge.Set =
.all) -> some View


    /// Disables capabilities of the
Photos picker.
    ///
```

```
    /// - Parameters:
    ///     - disabledCapabilities: One
or more of the available capabilities.
    /// - Returns: A Photos picker with
specified capabilities that are disabled.
    nonisolated public func
photosPickerDisabledCapabilities(_
disabledCapabilities:
PHPickerCapabilities) -> some View

}

// Available when SwiftUI is imported
with PhotosUI
@available(iOS 16.0, macOS 13.0, watchOS
9.0, *)
@available(tvOS, unavailable)
extension View {

    /// Presents a Photos picker that
selects a `PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - isPresented: The binding to
whether the Photos picker should be
shown.
    ///     - selection: The item being
shown and selected in the Photos picker.
```

```
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    nonisolated public func
photosPicker(isPresented: Binding<Bool>,
selection: Binding<PhotosPickerItem?>,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic) -> some View


    /// Presents a Photos picker that
selects a collection of
`PhotosPickerItem`.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - isPresented: The binding to
whether the Photos picker should be
shown.
    ///     - selection: All items being
```

```
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    nonisolated public func
photosPicker(isPresented: Binding<Bool>,
selection: Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic) -> some View


    /// Presents a Photos picker that
selects a `PhotosPickerItem` from a given
```

```
photo library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - isPresented: The binding to
whether the Photos picker should be
shown.
    ///     - selection: The item being
shown and selected in the Photos picker.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of the
selected item. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    @available(watchOS, unavailable)
    nonisolated public func
photosPicker(isPresented: Binding<Bool>,
selection: Binding<PhotosPickerItem?>,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
```

```
PHPhotoLibrary) -> some View


    /// Presents a Photos picker that
selects a collection of
`PhotosPickerItem` from a given photo
library.
    ///
    /// The user explicitly grants access
only to items they choose, so photo
library access authorization is not
needed.
    ///
    /// - Parameters:
    ///     - isPresented: The binding to
whether the Photos picker should be
shown.
    ///     - selection: All items being
shown and selected in the Photos picker.
    ///     - maxSelectionCount: The
maximum number of items that can be
selected. Default is `nil`. Setting it to
`nil` means maximum supported by the
system.
    ///     - selectionBehavior: The
selection behavior of the Photos picker.
Default is `.default`.
    ///     - filter: Types of items that
can be shown. Default is `nil`. Setting
it to `nil` means all supported types can
be shown.
    ///     - preferredItemEncoding: The
encoding disambiguation policy of
```

```
selected items. Default is `.automatic`.
Setting it to `.automatic` means the best
encoding determined by the system will be
used.
    ///     - photoLibrary: The photo
library to choose from.
    @available(watchOS, unavailable)
    nonisolated public func
photosPicker(isPresented: Binding<Bool>,
selection: Binding<[PhotosPickerItem]>,
maxSelectionCount: Int? = nil,
selectionBehavior:
PhotosPickerSelectionBehavior = .default,
matching filter: PHPickerFilter? = nil,
preferredItemEncoding:
PhotosPickerItem.EncodingDisambiguationPo
licy = .automatic, photoLibrary:
PHPhotoLibrary) -> some View

}

// Available when SwiftUI is imported
with PhotosUI
@available(iOS 16.0, macOS 13.0, *)
@available(watchOS, unavailable)
@available(tvOS, unavailable)
extension PHLivePhoto : Transferable {

    /// The representation used to import
and export the item.
    ///
    /// A ``transferRepresentation`` can
contain multiple representations
```

```
    /// for different content types.
    public static var
transferRepresentation: some
TransferRepresentation { get }

    /// The type of the representation
used to import and export the item.
    ///
    /// Swift infers this type from the
return value of the
    /// ``transferRepresentation``
property.
    @available(iOS 16.0, macOS 13.0, *)
    @available(tvOS, unavailable)
    @available(watchOS, unavailable)
    public typealias Representation =
some TransferRepresentation
}
```