```
import Combine
import Foundation
import UniformTypeIdentifiers
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims
import os
import os.log

//! Project version number for
CoreTransferable.
public var CoreTransferableVersionNumber:
Double

/// A transfer representation for types
that participate in Swift's protocols for
encoding and decoding.
///
///     struct Todo: Codable,
Transferable {
///         var text: String
///         var isDone = false
///
///         static var
transferRepresentation: some
TransferRepresentation {
///
CodableRepresentation(contentType: .todo)
///         }
///     }
///
///      extension UTType {
///          static var todo: UTType
```

```
{ UTType(exportedAs:
"com.example.todo") }
///     }
///
/// > Important: If your app declares
custom uniform type identifiers,
/// include corresponding entries in the
app's `Info.plist`.
/// For more information, see
<doc://com.apple.documentation/documentat
ion/uniformtypeidentifiers/
defining_file_and_data_types_for_your_app
>.
///
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct CodableRepresentation<Item,
Encoder, Decoder> :
TransferRepresentation, Sendable where
Item : Transferable, Item : Decodable,
Item : Encodable, Encoder :
TopLevelEncoder, Decoder :
TopLevelDecoder, Encoder.Output == Data,
Decoder.Input == Data {

    /// Creates a transfer representation
for a given type and type identifier.
    ///
    /// This initializer uses JSON for
encoding and decoding.
    ///
    /// - Parameters:
    ///   - itemType: The concrete type
```

```
of the item that's being transferred.
    ///   - contentType: A uniform type
identifier that best describes the item.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    public init(for itemType: Item.Type =
Item.self, contentType: UTType) where
Encoder == JSONEncoder, Decoder ==
JSONDecoder

    /// Creates a transfer representation
for a given type with the encoder and
decoder you supply.
    ///
    /// - Parameters:
    ///   - itemType: The concrete type
of the item that's being transported.
    ///   - contentType: A uniform type
identifier that best describes the item.
    ///   - encoder: An instance of a
type that can convert the item being
transferred
    ///   into binary data with a
specific structure.
    ///   - decoder: An instance of a
type that can convert specifically
structured
    ///   binary data into the item being
transferred.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    public init(for itemType: Item.Type =
Item.self, contentType: UTType, encoder:
```

```
Encoder, decoder: Decoder)

    /// The transfer representation for
the item.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Body = Never
}

/// A transfer representation for types
that provide their own binary data
conversion.
///
/// Use this transfer representation if
your model is stored in memory.
/// For example, a drawing app might have
a notion of a *layer*
/// that can be converted to and from a
custom binary data format and
/// also converted to the PNG image type:
///
///     struct ImageDocumentLayer {
///         init(data: Data) throws
///         func data() -> Data
///         func pngData() -> Data
///     }
///
/// You can provide multiple transfer
representations for a model type,
/// even if the transfer representation
types are the same.
/// The following shows the
`ImageDocumentLayer` structure
```

```
/// conforming to `Transferable` with two
``DataRepresentation`` instances
/// composed together:
///
///     extension ImageDocumentLayer:
Transferable {
///         static var
transferRepresentation: some
TransferRepresentation {
///
DataRepresentation(contentType: .layer) {
layer in
///                     layer.data()
///                 } importing: { data
in
///                     try
ImageDocumentLayer(data: data)
///                 }
///
DataRepresentation(exportedContentType: .
png) { layer in
///                 layer.pngData()
///             }
///         }
///     }
///
/// The example drawing app's custom
transfer representation comes first
/// so that apps that know about the
custom transfer representation can use
it.
/// The second transfer representation
offers export compatibility with other
```

```
apps
/// that work with PNG images.
///
/// > Tip: If a type conforms to
`Codable`, ``CodableRepresentation``
might be
/// a more convenient choice than
``DataRepresentation``.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct DataRepresentation<Item> :
TransferRepresentation where Item :
Transferable {

    /// Creates a representation that
allows transporting an item as binary
data.
    ///
    /// - Parameters:
    ///   - contentType: A uniform type
identifier that best describes the item.
    ///   - exporting: A closure that
provides a data representation of the
given item.
    ///   - importing: A closure that
instantiates the item with given binary
data.
    public init(contentType: UTType,
exporting: @escaping @Sendable (Item)
async throws -> Data, importing:
@escaping @Sendable (Data) async throws
-> Item)
```

```
    /// Creates a representation that
allows exporting an item as binary data.
    ///
    /// - Parameters:
    ///   - exportedContentType: A
uniform type identifier that best
describes the item.
    ///   - exporting: A closure that
provides a data representation of the
given item.
    public init(exportedContentType:
UTType, exporting: @escaping @Sendable
(Item) async throws -> Data)

    /// Creates a representation that
allows importing an item as binary data.
    ///
    /// - Parameters:
    ///   - importedContentType: A
uniform type identifier that best
describes the item.
    ///   - importing: A closure that
instantiates the item with given binary
data
    public init(importedContentType:
UTType, importing: @escaping @Sendable
(Data) async throws -> Item)

    /// The transfer representation for
the item.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Body = Never
```

```
}

/// A transfer representation for types
that transfer as a file URL.
///
/// Use a ``FileRepresentation`` for
transferring types
/// that involve a large amount of data.
/// For example, if your app defines a
`Movie` type that could represent a
lengthy video,
/// use a `FileRepresentation` instance
/// to transfer the video data to another
app or process.
///
///     struct Movie: Transferable {
///         let url: URL
///         static var
transferRepresentation: some
TransferRepresentation {
///
FileRepresentation(contentType: .mpeg4Mov
ie) { movie in
///
SentTransferredFile($0.url)
///                 } importing:
{ received in
///                     let copy: URL =
URL(fileURLWithPath: "<#...#>")
///                     try
FileManager.default.copyItem(at:
received.file, to: copy)
///                     return
```

```
Self.init(url: copy) }
///         }
///     }
/// It's efficient to pass such data
around as a file and the receiver
/// loads it into memory only if it's
required.
///
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct FileRepresentation<Item> :
TransferRepresentation where Item :
Transferable {

    /// Creates a transfer representation
for importing and exporting
    /// transferable items as files.
    ///
    /// - Parameters:
    ///   - contentType: A uniform type
identifier that best describes the item.
    ///   - shouldAttemptToOpenInPlace: A
Boolean value that
    ///   indicates whether the receiver
gains access to the original item on disk
    ///   and can edit it,
    ///   or to a copy made by the
system.
    ///   - exporting: A closure that
provides a file representation of the
given item.
    ///   - importing: A closure that
instantiates the item with given file
```

```
promise.
    /// The file referred to by the
    /// ``ReceivedTransferredFile/file``
property of the
    /// ``ReceivedTransferredFile``
instance
    /// is only guaranteed to exist
within the `importing` closure. If you
need the file
    /// to be around for a longer period,
make a copy in the `importing` closure.
    public init(contentType: UTType,
shouldAttemptToOpenInPlace: Bool = false,
exporting: @escaping @Sendable (Item)
async throws -> SentTransferredFile,
importing: @escaping @Sendable
(ReceivedTransferredFile) async throws ->
Item)

    /// Creates a transfer representation
for exporting transferable items as
files.
    ///
    /// - Parameters:
    ///   - exportedContentType: A
uniform type identifier for the file
`URL`,
    ///   returned by the `exporting`
closure.
    ///   - shouldAllowToOpenInPlace: A
Boolean value that indicates whether
    ///   the receiver can try to gain
access to the original item on disk
```

```
    ///   and can edit it.
    ///   If `false`, the receiver only
has access to a copy of the file
    ///   made by the system.
    ///   - exporting: A closure that
provides a file representation of the
given item.
    public init(exportedContentType:
UTType, shouldAllowToOpenInPlace: Bool =
false, exporting: @escaping @Sendable
(Item) async throws ->
SentTransferredFile)

    /// Creates a transfer representation
for importing transferable items as
files.
    ///
    /// - Parameters:
    ///   - importedContentType: A
uniform type identifier for the file
promise,
    ///   returned by the `exporting`
closure.
    ///   - shouldAttemptToOpenInPlace: A
Boolean value that indicates whether
    ///   the receiver wants to gain
access to the original item on disk
    ///   and can edit it.
    ///   If `false`, the receiver only
has access to a copy of the file
    ///   made by the system.
    ///   - importing: A closure that
creates the item with given file promise.
```

```
    /// The file referred to by the
`file` property of the
`ReceivedTransferredFile`
    /// is only guaranteed to exist
within the `importing` closure. If you
need the file
    /// to be around for a longer period,
make a copy in the `importing` closure.
    public init(importedContentType:
UTType, shouldAttemptToOpenInPlace: Bool
= false, importing: @escaping @Sendable
(ReceivedTransferredFile) async throws ->
Item)

    /// The transfer representation for
the item.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Body = Never
}

/// A transfer representation that uses
another type's transfer representation
/// as its own.
///
/// Use this representation to rely on an
existing transfer representation
/// that's suitable for the type.
/// For example,  a `Note` type might use
the
///
<doc://com.apple.documentation/documentat
ion/Swift/String> structure's
```

```
/// built-in ``Transferable`` conformance
---
/// a plain text representation --- so it
can be pasted into any text editor:
///
///     struct Note: Transferable {
///         var body: String
///
///         static var
transferRepresentation: some
TransferRepresentation {
///
ProxyRepresentation(\.body)
///         }
///     }
///
/// `ProxyRepresentation` makes it easy
to provide alternative representations
/// for receivers that don't support the
preferred custom format.
///
///      struct Todo: Transferable,
Codable {
///         var text: String
///         var isDone = false
///
///         static var
transferRepresentation: some
TransferRepresentation {
///
CodableRepresentation(contentType: .todo)
///
ProxyRepresentation(\.text)
```

```
///         }
///     }
///
///      extension UTType {
///          static var todo: UTType
{ UTType(exportedAs:
"com.example.todo") }
///      }
///
/// Write the order of the
representations in the
`transferRepresentation` property
/// from more preferred to less
preferred. In the previous example, if
the receiver knows
/// about the custom `com.example.todo`
content type, it will receive that custom
content type.
/// Using a `ProxyRepresentation` as the
alternative lets people paste
/// the to-do item in any text editor
that doesn't support the
`com.example.todo`
/// content type but works with text
formats.
///
/// `ProxyRepresentation` is a
convenience, and its evaluation isn't
supposed
/// to be calculation-heavy. Don't
perform long-running work
/// in `exporting` and `importing`
closures. They shouldn't contain
```

```
/// network requests, file operations, or
other potentially time-consuming tasks
/// as they can cause delays during
operations with `Transferable` items.
///
/// Use
<doc://com.apple.documentation/documentat
ion/coretransferable/filerepresentation>
/// or
<doc://com.apple.documentation/documentat
ion/coretransferable/datarepresentation>
/// to read and write files or for other
lengthy tasks.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct ProxyRepresentation<Item,
ProxyRepresentation> :
TransferRepresentation where Item :
Transferable, ProxyRepresentation :
Transferable {

    /// Creates a transfer representation
that's imported by proxy through
    /// another transfer representation.
    ///
    /// - Parameters:
    ///   - importing: A closure that
converts the chosen representation into
    ///   the transported item.
    public init(importing: @escaping
@Sendable (ProxyRepresentation) throws ->
Item)
```

```
    /// Creates a transfer representation
that's imported by proxy through
    /// another transfer representation.
    ///
    /// - Parameters:
    ///   - importing: A closure that
converts the chosen representation into
    ///   the transported item.
    @available(iOS 16.1, macOS 13.0, tvOS
16.1, watchOS 9.1, *)
    public init(importing: @escaping
@Sendable (ProxyRepresentation) async
throws -> Item)

    /// Creates a transfer representation
that's exported by proxy through
    /// another transfer representation.
    ///
    /// - Parameters:
    ///   - exporting: A closure that
converts the item into
    ///   desired representation.
    public init(exporting: @escaping
@Sendable (Item) throws ->
ProxyRepresentation)

    /// Creates a transfer representation
that's exported by proxy through
    /// another transfer representation.
    ///
    /// - Parameters:
    ///   - exporting: A closure that
converts the item into
```

```
    ///   desired representation.
    @available(iOS, introduced: 16.1,
deprecated: 17.0, message: "A synchronous
exporter should be used instead.")
    @available(macOS, introduced: 13.0,
deprecated: 14.0, message: "A synchronous
exporter should be used instead.")
    @available(tvOS, introduced: 16.1,
deprecated: 17.0, message: "A synchronous
exporter should be used instead.")
    @available(watchOS, introduced: 9.1,
deprecated: 10.0, message: "A synchronous
exporter should be used instead.")
    public init(exporting: @escaping
@Sendable (Item) async throws ->
ProxyRepresentation)

    /// Creates a transfer representation
that's imported and exported
    /// by proxy through another transfer
representation.
    ///
    /// - Parameters:
    ///   - exporting: A closure that
converts the item into
    ///   desired representation.
    ///   - importing: A closure that
converts the chosen representation
    ///   back into the transported item.
    public init(exporting: @escaping
@Sendable (Item) throws ->
ProxyRepresentation, importing: @escaping
@Sendable (ProxyRepresentation) throws ->
```

```
Item)

    /// Creates a transfer representation
that's imported and exported
    /// by proxy through another transfer
representation.
    ///
    /// - Parameters:
    ///   - exporting: A closure that
converts the item into
    ///   desired representation.
    ///   - importing: A closure that
converts the chosen representation
    ///   back into the transported item.
    @available(iOS, introduced: 16.1,
deprecated: 17.0, message: "A synchronous
exporter should be used instead.")
    @available(macOS, introduced: 13.0,
deprecated: 14.0, message: "A synchronous
exporter should be used instead.")
    @available(tvOS, introduced: 16.1,
deprecated: 17.0, message: "A synchronous
exporter should be used instead.")
    @available(watchOS, introduced: 9.1,
deprecated: 10.0, message: "A synchronous
exporter should be used instead.")
    public init(exporting: @escaping
@Sendable (Item) async throws ->
ProxyRepresentation, importing: @escaping
@Sendable (ProxyRepresentation) async
throws -> Item)

    /// Creates a transfer representation
```

```
that's imported and exported
    /// by proxy through another transfer
representation.
    ///
    /// - Parameters:
    ///   - exporting: A closure that
converts the item into
    ///   desired representation.
    ///   - importing: A closure that
converts the chosen representation
    ///   back into the transported item.
    @available(iOS 17.2, macOS 14.2, tvOS
17.2, watchOS 10.2, *)
    public init(exporting: @escaping
@Sendable (Item) throws ->
ProxyRepresentation, importing: @escaping
@Sendable (ProxyRepresentation) async
throws -> Item)

    /// The transfer representation for
the item.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Body = Never
}

/// A description of a file from the
perspective of the receiver.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct ReceivedTransferredFile :
Sendable {
```

```
    /// The received file on disk.
    public let file: URL

    /// A Boolean value that indicates
whether the file's URL
    /// points to the original file
provided by the sender
    /// or to a copy.
    public let isOriginalFile: Bool
}

/// A description of a file from the
perspective of the sender.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct SentTransferredFile :
Sendable {

    /// A URL that describes the location
of the file.
    public let file: URL

    /// A Boolean value that indicates
whether
    /// the receiver can read and write
the original file.
    /// When set to `false`, the receiver
can only gain access to a copy of the
file.
    public let
allowAccessingOriginalFile: Bool

    /// Creates a description of a file
```

```
from the perspective of the sender.
    ///
    /// - Parameters:
    ///   - file: A URL that describes
the location of the file.
    ///   - allowAccessingOriginalFile: A
Boolean value that indicates whether
    /// the receiver can read and write
the original file.
    /// When set to `false`, the receiver
can only gain access to a copy of the
file.
    public init(_ file: URL,
allowAccessingOriginalFile: Bool = false)
}

/// A declarative description of the
process of importing and exporting a
transferable item.
///
/// Combine multiple existing transfer
representations
/// to compose a single transfer
representation that describes
/// how to transfer an item in multiple
scenarios.
///
/// The following shows a `Greeting` type
that transfers both as a `Codable` type
/// and by proxy through its `message`
string.
///
///     import UniformTypeIdentifiers
```

```
///
///     struct Greeting: Codable,
Transferable {
///         let message: String
///         var displayInAllCaps: Bool =
false
///
///         static var
transferRepresentation: some
TransferRepresentation {
///
CodableRepresentation(contentType: .greet
ing)
///
ProxyRepresentation(exporting: \.message)
///         }
///     }
///
///     extension UTType {
///         static var greeting: UTType {
.init(exportedAs: "com.example.greeting")
}
///     }
///
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public protocol
TransferRepresentation<Item> : Sendable {

    /// The type of the item that's being
transferred.
    associatedtype Item : Transferable
```

```
    /// The transfer representation for
the item.
    associatedtype Body :
TransferRepresentation

    /// A builder expression that
describes the process of importing and
exporting an item.
    ///
    /// Combine multiple existing
transfer representations
    /// to compose a single transfer
representation that describes
    /// how to transfer an item in
multiple scenarios.
    ///
    ///     struct
CombinedRepresentation:
TransferRepresentation {
    ///         var body: some
TransferRepresentation {
    ///
DataRepresentation(...)
    ///
FileRepresentation(...)
    ///         }
    ///     }
    ///

@TransferRepresentationBuilder<Self.Item>
var body: Self.Body { get }
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension TransferRepresentation {

    /// Specifies the kinds of apps and
processes that can see an item in
transit.
    ///
    /// - Parameters:
    ///   - visibility: The visibility
level.
    public func visibility(_ visibility:
TransferRepresentationVisibility) -> some
TransferRepresentation<Self.Item>

}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension TransferRepresentation {

    /// Prevents the system from
exporting an item if it does not meet the
supplied condition.
    ///
    /// Some instances of a model type
may have state-dependent conditions that
make them
    /// unsuitable for export. For
example, an `Archive` structure that
supports
    /// a comma-separated text
representation only when it has
```

```
compatible content:
    ///
    ///     struct Archive {
    ///         var supportsCSV: Bool
    ///         func csvData() -> Data
    ///         init(csvData: Data)
    ///     }
    ///
    ///     extension Archive:
Transferable {
    ///         static var
transferRepresentation: some
TransferRepresentation {
    ///
DataRepresentation(contentType: .commaSep
aratedText) { archive in
    ///                 archive.csvData()
    ///             } importing: { data
in Archive(csvData: data) }
    ///                 .exportingConditi
on { archive in archive.supportsCSV }
    ///         }
    ///     }
    ///
    /// - Parameters:
    ///   - condition: A closure that
determines whether the item is
exportable.
    ///   Don't perform long-running work
in this closure.
    ///   It shouldn't contain network
requests, file operations,
    ///   or other potentially time-
```

```
consuming tasks as they
    ///   can cause delays during
operations with `Transferable` items.
    public func exportingCondition(_
condition: @escaping @Sendable
(Self.Item) -> Bool) ->
_ConditionalTransferRepresentation<Self>
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension TransferRepresentation {

    /// Provides a filename to use if the
receiver chooses to write the item to
disk.
    ///
    /// Any transfer representation can
be written to disk.
    ///
    ///      extension
ImageDocumentLayer: Transferable {
    ///           static var
transferRepresentation: some
TransferRepresentation {
    ///
DataRepresentation(contentType: .layer) {
layer in
    ///                     layer.data()
    ///                     } importing:
{ data in
    ///                         try
ImageDocumentLayer(data: data)
```

```
    ///                     }
    ///                     .suggestedFileNa
me("Layer.exampleLayer")
    ///
DataRepresentation(exportedContentType: .
png) { layer in
    ///                     layer.pngData()
    ///                 }
    ///                 .suggestedFileName("
Layer.png")
    ///             }
    ///         }
    ///
    /// The .exampleLayer filename
extension above should match
    /// the extension for the `layer`
content type,
    /// which you declare in your app's
`Info.plist` file.
    ///
    /// - Parameters:
    ///   - fileName: The suggested
filename including the filename
extension.
    ///     If several suggested file
names are specified on an item, only the
last one will be used.
    public func suggestedFileName(_
fileName: String) -> some
TransferRepresentation<Self.Item>


    /// Provides a filename to use if the
```

```
receiver chooses to write the item to
disk.
    ///
    /// Any transfer representation can
be written to disk.
    ///
    ///     struct Note: Transferable {
    ///         var title: String
    ///         var body: String
    ///         static var
transferRepresentation: some
TransferRepresentation {
    ///
ProxyRepresentation(exporting: \.body)
    ///                     .suggestedFileNam
e { $0.title + ".txt" }
    ///             }
    ///     }
    ///
    /// - Parameters:
    ///   - fileName: The optional
closure that returns the suggested
filename
    ///     including the filename
extension.
    ///     If several suggested file
names are specified on an item, only the
last one will be used.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func suggestedFileName(_
fileName: @escaping @Sendable (Self.Item)
-> String?) -> some
```

```
TransferRepresentation<Self.Item>

}

/// Creates a transfer representation by
composing existing transfer
representations.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
@resultBuilder public struct
TransferRepresentationBuilder<Item> where
Item : Transferable {

    /// Builds an encodable and decodable
transfer representation from an
expression.
    public static func
buildExpression<Encoder, Decoder>(_
content: CodableRepresentation<Item,
Encoder, Decoder>) ->
CodableRepresentation<Item, Encoder,
Decoder> where Item : Decodable, Item :
Encodable, Encoder : TopLevelEncoder,
Decoder : TopLevelDecoder, Encoder.Output
== Data, Decoder.Input == Data

    /// Builds a transfer representation
from an expression.
    public static func
buildExpression<R>(_ content: R) -> R
where Item == R.Item, R :
TransferRepresentation
```

```
    /// Passes a single transfer
representation to the builder unmodified.
    public static func
buildBlock<Content>(_ content: Content)
-> Content where Item == Content.Item,
Content : TransferRepresentation

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1,
C2>(_ content1: C1, _ content2: C2) ->
TupleTransferRepresentation<Item, (C1,
C2)> where Item == C1.Item, C1 :
TransferRepresentation, C2 :
TransferRepresentation, C1.Item ==
C2.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1, C2,
C3>(_ content1: C1, _ content2: C2, _
content3: C3) ->
TupleTransferRepresentation<Item, (C1,
C2, C3)> where Item == C1.Item, C1 :
TransferRepresentation, C2 :
TransferRepresentation, C3 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item

    /// Combines multiple transfer
representations into a single transfer
```

```
representation.
    public static func buildBlock<C1, C2,
C3, C4>(_ content1: C1, _ content2: C2, _
content3: C3, _ content4: C4) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4)> where Item == C1.Item, C1 :
TransferRepresentation, C2 :
TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1, C2,
C3, C4, C5>(_ content1: C1, _ content2:
C2, _ content3: C3, _ content4: C4, _
content5: C5) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4, C5)> where Item == C1.Item,
C1 : TransferRepresentation, C2 :
TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C5 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item, C4.Item == C5.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
```

```
    public static func buildBlock<C1, C2,
C3, C4, C5, C6>(_ content1: C1, _
content2: C2, _ content3: C3, _ content4:
C4, _ content5: C5, _ content6: C6) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4, C5, C6)> where Item ==
C1.Item, C1 : TransferRepresentation,
C2 : TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C5 :
TransferRepresentation, C6 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item, C4.Item == C5.Item, C5.Item ==
C6.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1, C2,
C3, C4, C5, C6, C7>(_ content1: C1, _
content2: C2, _ content3: C3, _ content4:
C4, _ content5: C5, _ content6: C6, _
content7: C7) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4, C5, C6, C7)> where Item ==
C1.Item, C1 : TransferRepresentation,
C2 : TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C5 :
TransferRepresentation, C6 :
TransferRepresentation, C7 :
TransferRepresentation, C1.Item ==
```

```
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item, C4.Item == C5.Item, C5.Item ==
C6.Item, C6.Item == C7.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1, C2,
C3, C4, C5, C6, C7, C8>(_ content1: C1, _
content2: C2, _ content3: C3, _ content4:
C4, _ content5: C5, _ content6: C6, _
content7: C7, _ content8: C8) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4, C5, C6, C7, C8)> where Item
== C1.Item, C1 : TransferRepresentation,
C2 : TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C5 :
TransferRepresentation, C6 :
TransferRepresentation, C7 :
TransferRepresentation, C8 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item, C4.Item == C5.Item, C5.Item ==
C6.Item, C6.Item == C7.Item, C7.Item ==
C8.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1, C2,
C3, C4, C5, C6, C7, C8, C9>(_ content1:
C1, _ content2: C2, _ content3: C3, _
```

```
content4: C4, _ content5: C5, _ content6:
C6, _ content7: C7, _ content8: C8, _
content9: C9) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4, C5, C6, C7, C8, C9)> where
Item == C1.Item, C1 :
TransferRepresentation, C2 :
TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C5 :
TransferRepresentation, C6 :
TransferRepresentation, C7 :
TransferRepresentation, C8 :
TransferRepresentation, C9 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item, C4.Item == C5.Item, C5.Item ==
C6.Item, C6.Item == C7.Item, C7.Item ==
C8.Item, C8.Item == C9.Item

    /// Combines multiple transfer
representations into a single transfer
representation.
    public static func buildBlock<C1, C2,
C3, C4, C5, C6, C7, C8, C9, C10>(_
content1: C1, _ content2: C2, _ content3:
C3, _ content4: C4, _ content5: C5, _
content6: C6, _ content7: C7, _ content8:
C8, _ content9: C9, _ content10: C10) ->
TupleTransferRepresentation<Item, (C1,
C2, C3, C4, C5, C6, C7, C8, C9, C10)>
where Item == C1.Item, C1 :
TransferRepresentation, C2 :
```

```
TransferRepresentation, C3 :
TransferRepresentation, C4 :
TransferRepresentation, C5 :
TransferRepresentation, C6 :
TransferRepresentation, C7 :
TransferRepresentation, C8 :
TransferRepresentation, C9 :
TransferRepresentation, C10 :
TransferRepresentation, C1.Item ==
C2.Item, C2.Item == C3.Item, C3.Item ==
C4.Item, C4.Item == C5.Item, C5.Item ==
C6.Item, C6.Item == C7.Item, C7.Item ==
C8.Item, C8.Item == C9.Item, C9.Item ==
C10.Item
}

/// The visibility levels that specify
the kinds of apps and processes
/// that can see an item in transit.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct
TransferRepresentationVisibility :
Sendable, Equatable {

    /// The visibility level that
specifies that any app or process can
access the item.
    public static let all:
TransferRepresentationVisibility

    /// The visibility level that
specifies that the item is visible only
```

```
    /// to macOS apps in the same App
Group.
    @available(iOS, unavailable)
    @available(tvOS, unavailable)
    @available(watchOS, unavailable)
    public static let group:
TransferRepresentationVisibility

    /// The visibility level that
specifies that the item is visible only
    /// within the app that's the source
of the item.
    public static let ownProcess:
TransferRepresentationVisibility

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

/// A protocol that describes how a type
```

```
interacts with transport APIs
/// such as drag and drop or copy and
paste.
///
/// To conform to the ``Transferable``
protocol,
/// implement the
``transferRepresentation`` property.
/// For example, an image editing app's
layer type might
/// conform to `Transferable` to let
people drag and drop image layers
/// to reorder them within a document.
///
///     struct ImageDocumentLayer {
///         init(data: Data)
///         func data() -> Data
///         func pngData() -> Data
///     }
///
/// The following shows how you can
extend `ImageDocumentLayer`  to
/// conform to `Transferable`:
///
///     extension ImageDocumentLayer:
Transferable {
///         static var
transferRepresentation: some
TransferRepresentation {
///
DataRepresentation(contentType: .layer) {
layer in
///                 layer.data()
```

```
///             } importing: { data in
///
ImageDocumentLayer(data: data)
///             }
///
DataRepresentation(exportedContentType: .
png) { layer in
///                 layer.pngData()
///             }
///         }
///     }
///
/// When people drag and drop a layer
within the app or onto another app
/// that recognizes the custom `layer`
content type,
/// the app uses the first
representation.
/// When people drag and drop the layer
onto a different image editor,
/// it's likely that the editor
recognizes the PNG file type.
/// The second transfer representation
adds support for PNG files.
///
/// The following declares the custom
`layer` uniform type identifier:
///
///     extension UTType {
///         static var layer: UTType
{ UTType(exportedAs: "com.example.layer")
}
///     }
```

```
///
/// > Important: If your app declares
custom uniform type identifiers,
/// include corresponding entries in the
app's `Info.plist`.
/// For more information, see
<doc://com.apple.documentation/documentat
ion/uniformtypeidentifiers/
defining_file_and_data_types_for_your_app
>.
///
/// If one of your existing types
conforms to
<doc://com.apple.documentation/documentat
ion/Swift/Codable>,
/// `Transferable` automatically handles
conversion to and from `Data`.
/// The following declares a simple
`Note` structure that's `Codable`
/// and an extension to make it
`Transferable`:
///
///     struct Note: Codable {
///         let title: String
///         let body: String
///     }
///
///     extension Note: Transferable {
///         static var
transferRepresentation: some
TransferRepresentation {
///
CodableRepresentation(contentType: .note)
```

```
///         }
///     }
///
/// To ensure compatibility with other
apps that don't know about
/// the custom `note` type identifier,
/// the following adds an additional
transfer representation
/// that converts the note to text.
///
///     extension Note: Transferable {
///         static var
transferRepresentation: some
TransferRepresentation {
///
CodableRepresentation(contentType: .note)
///
ProxyRepresentation(\.title)
///         }
///     }
/// The order of the representations in
the transfer representation matters;
/// place the representation that most
accurately represents your type first,
/// followed by a sequence of more
compatible
/// but less preferable representations.
///
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public protocol Transferable {

    /// The type of the representation
```

```
used to import and export the item.
    ///
    /// Swift infers this type from the
return value of the
    /// ``transferRepresentation``
property.
    associatedtype Representation :
TransferRepresentation

    /// The representation used to import
and export the item.
    ///
    /// A ``transferRepresentation`` can
contain multiple representations
    /// for different content types.
    @TransferRepresentationBuilder<Self>
static var transferRepresentation:
Self.Representation { get }
}

@available(iOS 18.2, macOS 15.2, tvOS
18.2, watchOS 11.2, visionOS 2.2, *)
extension Transferable {

    /// The types that the instance of a
`Transferable` is able to provide
    /// a representation for.
    public static func
exportedContentTypes(visibility:
TransferRepresentationVisibility = .all)
-> [UTType]

    /// Content types statically
```

```
supported by the `Transferable`
conformance of the type
    /// for import (like drop or paste).
    ///
    /// For example, if you have a type
that conforms to ``Transferable``
    /// and can be represented as
<doc://com.apple.documentation/documentat
ion/SwiftUI/Image>
    /// and you need to know if it can be
instantiated with a JPEG file, you can
check against
    /// `importedContentTypes`:
    ///
    ///     struct Icon: Transferable {
    ///         var image: Image
    ///         static var
transferRepresentation: some
TransferRepresentation {
    ///
ProxyRepresentation(\.image)
    ///         }
    ///     }
    ///
    ///     let isJPEGSupported =
Image.importedContentTypes().contains(.jp
eg)
    ///
    /// The default implementation of
this function is available to all types
that conform
    /// to `Transferable` protocol.
    public static func
```

```
importedContentTypes() -> [UTType]

    /// Content types supported by a
given value's `Transferable` conformance
    /// for export (like drag or copy).
    ///
    /// Returns a list of all content
types available for export in the type's
    /// ``Transferable`` conformance,
except for those
    /// that are not supported by this
specific value. For example, if an
``exportingCondition`` for some
    /// representation evaluates to
`false`, this representation isn't
included.
    ///
    /// The default implementation of
this function is available to all types
that conform
    /// to `Transferable` protocol.
    /// - Parameters:
    ///   - visibility: Desired
visibility level for the returned content
types. Defaults to `.all`.
    public func exportedContentTypes(_
visibility:
TransferRepresentationVisibility = .all)
-> [UTType]

    /// Content types supported by a
given value's `Transferable` conformance
    /// for import (like drop or paste).
```

```
    ///
    /// Returns a list of all content
types available for import.
    /// The default implementation of
this function is available
    /// to all types that conform to
``Transferable`` protocol.
    public func importedContentTypes() ->
[UTType]

    /// Using the type's `Transferable`
conformance implementation, instantiates
a
    /// value from the given file.
    ///
    /// The default implementation of
this initializer is available to all
types that conform
    /// to ``Transferable`` protocol.
    ///
    /// - Parameters:
    ///   - file: A URL to a file on
disk.
    ///   - contentType: An optional
content type for creating a value.
    ///   If a value is not provided, the
initializer tries to infer it from
    ///   the file extension or its
metadata. If the content type is still
unknown,
    ///   the framework calls the first
transfer representation with this URL.
    ///   If the item isn't imported
```

```
successfully, the framework calls the
second representation and
    ///   so on.
    ///
    public init(importing file: URL,
contentType: UTType?) async throws

    /// Using the type's `Transferable`
conformance implementation,
    /// exports a value by writing it to
a provided destination directory.
    ///
    /// If the ``Transferable`` is not
backed by a file,
    /// this will write the data to
specified destination.
    /// This function uses the first
representation provided for a given type
    /// in `static var
transferRepresentation` requirement of
the `Transferable` protocol
    /// or in the `body` of a custom
`TransferRepresentation`.
    ///
    /// The default implementation of
this function is available to all types
that conform
    /// to `Transferable` protocol.
    /// - Parameters:
    ///   - destinationDirectory: A
directory to write the file to.
    ///   - contentType: A content type
of the requested file.
```

```
    /// If `nil`, the first transfer
representation is be used.
    /// - Returns: A URL of the created
file. The file is owned by the
application,
    /// and it is responsible for
removing it when the file is not needed
anymore.
    public func export(to
destinationDirectory: URL, contentType:
UTType?) async throws -> URL

    /// Using the type's `Transferable`
conformance implementation,
    /// exports a value by writing it to
disk and removes when not needed.
    ///
    /// This converts a ``Transferable``
item into a temporary file, and
    /// removes it after `fileHandler`
closure returns. The default
implementation
    /// of this function is available to
all types that conform
    /// to `Transferable` protocol.
    ///
    /// - Parameters:
    ///   - contentType: A content type
of the requested file.
    ///   If the content type is not
provided, CoreTransferable creates
    ///   a file from the first
`TransferRepresentation` that supports
```

```
export.
    ///   - fileHandler: A closure that
accepts a file URL as a parameter.
    /// The file is written to a
temporary destination and removed
    /// after the closure returns.
    public func
withExportedFile<Result>(contentType:
UTType?, fileHandler: (URL) async throws
-> Result) async throws -> Result

    /// A suggested filename of a
`Transferable` value.
    ///
    /// A filename for given item, or
`nil` if none specified.
    /// A name can be specified using
``TransferRepresentation.suggestedFileNam
e(_:)``.
    /// The default implementation of
this property is available to all types
that conform
    /// to ``Transferable`` protocol.
    public var suggestedFilename: String?
{ get }

    /// Using the type's `Transferable`
conformance implementation, instantiates
a
    /// value from given data.
    ///
    /// The default implementation of
this initializer is available to all
```

```
    /// types that conform to
``Transferable`` protocol.
    ///
    /// - Parameters:
    ///   - data: Binary data that can be
used to instantiate an item
    ///   - contentType: A content type
that corresponds
    ///   to the structure of the data.
If no content type is provided, the
framework
    ///   calls into every transfer
representation provided in the
implementation
    ///   of the ``Transferable``
conformance (`transferRepresentation`
static
    ///   property) until it finds one
that can handle the data.
    public init(importing data: Data,
contentType: UTType?) async throws

    /// Using the type's `Transferable`
conformance implementation,
    /// exports a value as binary data.
    ///
    /// If the ``Transferable`` is backed
by a file `URL`, this might cause loading
the
    /// entire file contents into memory.
This function uses
    /// the first representation that
conforms to the given content type
```

```
    /// in `static var
transferRepresentation` requirement of
the `Transferable` protocol
    /// or in the `var body` property of
a custom `TransferRepresentation`.
    ///
    /// The default implementation of
this function is available to all
    /// types that conform to
`Transferable` protocol.
    /// - Parameters:
    ///   - contentType: A content type
of the returned data.
    ///   If no content type is provided,
CoreTransferable generates data using
    ///   the first
`TransferRepresentation` in the
`Transferable` conformance that supports
export.
    public func exported(as contentType:
UTType?) async throws -> Data
}

/// A wrapper type for tuples that
contain transfer representations.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct
TupleTransferRepresentation<Item,
Value> : TransferRepresentation where
Item : Transferable, Value : Sendable {

    /// A builder expression that
```

```
describes the process of importing and
exporting an item.
    ///
    /// Combine multiple existing
transfer representations
    /// to compose a single transfer
representation that describes
    /// how to transfer an item in
multiple scenarios.
    ///
    ///     struct
CombinedRepresentation:
TransferRepresentation {
    ///         var body: some
TransferRepresentation {
    ///
DataRepresentation(...)
    ///
FileRepresentation(...)
    ///         }
    ///     }
    ///
    public var body: some
TransferRepresentation { get }

    /// The transfer representation for
the item.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Body = some
TransferRepresentation
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Never {

    /// The transfer representation for
the item.
    public typealias Body = Never

    /// A builder expression that
describes the process of importing and
exporting an item.
    ///
    /// Combine multiple existing
transfer representations
    /// to compose a single transfer
representation that describes
    /// how to transfer an item in
multiple scenarios.
    ///
    ///     struct
CombinedRepresentation:
TransferRepresentation {
    ///         var body: some
TransferRepresentation {
    ///
DataRepresentation(...)
    ///
FileRepresentation(...)
    ///         }
    ///     }
    ///
    public var body: Never { get }
}
```

```
/// To support returning `Never` from the
body of a
`PrimitiveTransferRepresentation`,
/// `Never`  also conforms to
`TransferRepresentation`.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Never : TransferRepresentation
{

    /// The type of the item that's being
transferred.
    public typealias Item = Never
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Never : Transferable {

    /// The type of the representation
used to import and export the item.
    ///
    /// Swift infers this type from the
return value of the
    /// ``transferRepresentation``
property.
    public typealias Representation =
Never

    /// The representation used to import
and export the item.
    ///
```

```
    /// A ``transferRepresentation`` can
contain multiple representations
    /// for different content types.
    public static var
transferRepresentation: Never { get }
}

@available(iOS 16.1, macOS 13.0, tvOS
16.1, watchOS 9.1, *)
extension AttributedString : Transferable
{

    /// The representation used to import
and export the item.
    ///
    /// A ``transferRepresentation`` can
contain multiple representations
    /// for different content types.
    @available(iOS 16.1, macOS 13.0, tvOS
16.1, watchOS 9.1, *)
    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 16.1, tvOS 16.1,
watchOS 9.1, macOS 13.0, *)
```

```
    public typealias Representation =
some TransferRepresentation
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Data : Transferable, @unchecked
Sendable {

    /// 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, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Representation =
some TransferRepresentation
}

@available(iOS 16.0, macOS 13.0, tvOS
```

```
16.0, watchOS 9.0, *)
extension String : 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, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Representation =
some TransferRepresentation
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension URL : Transferable, @unchecked
Sendable {

    /// 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, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Representation =
some TransferRepresentation
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension NSItemProvider {

    /// Registers every transfer
representation of given item with the
receiver.
    /// - Parameter transferable: The
item to register.
    public func register<T>(_
transferable: @autoclosure @escaping
@Sendable () -> T) where T : Transferable
```

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