Protocol

Transferable

A protocol that describes how a type interacts with transport APIs such as drag and drop or copy and paste.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | tvOS 16.0+ | visionOS 1.0+ | watchOS 9.0+

```
@preconcurrency
protocol Transferable : Sendable
```

Overview

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 { Data() }
    func pngData() -> Data { 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 let layer = 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 Defining file and data types for your app.

If one of your existing types conforms to `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.

```swift
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.

Topics

Implementing a transfer representation

`static var transferRepresentation: Self.Representation`

The representation used to import and export the item.

Required

`associatedtype Representation : TransferRepresentation`

The type of the representation used to import and export the item.

Required

Initializers

`init(importing: URL, contentType: UTType?) async throws`

Using the type's `Transferable` conformance implementation, instantiates a value from the given file.

`init(importing: Data, contentType: UTType?) async throws`

Using the type's `Transferable` conformance implementation, instantiates a value from given data.

Instance Properties

`var suggestedFilename: String?`

A suggested filename of a `Transferable` value.

Instance Methods

`func export(to: URL, contentType: UTType?) async throws -> URL`

> Using the type’s `Transferable` conformance implementation, exports a value by writing it to a provided destination directory.

`func exported(as: UTType?) async throws -> Data`

> Using the type’s `Transferable` conformance implementation, exports a value as binary data.

`func exportedContentTypes(TransferRepresentationVisibility) -> [UTType]`

> Content types supported by a given value’s `Transferable` conformance for export (like drag or copy).

`func importedContentTypes() -> [UTType]`

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

`func withExportedFile<Result>(contentType: UTType?, fileHandler: (URL) async throws -> Result) async throws -> Result`

> Using the type’s `Transferable` conformance implementation, exports a value by writing it to disk and removes when not needed.

Type Methods

`static func exportedContentTypes(visibility: TransferRepresentationVisibility) -> [UTType]`

> The types that the instance of a `Transferable` is able to provide a representation for.

`static func importedContentTypes() -> [UTType]`

> Content types statically supported by the `Transferable` conformance of the type for import (like drop or paste).

Relationships

Inherits From

`Sendable`, `SendableMetatype`

See Also

Essentials

`protocol TransferRepresentation`

A declarative description of the process of importing and exporting a transferable item.

Choosing a transfer representation for a model type

Define a custom representation for your data using a combination of built-in types.