Class

NSItemProvider

An item provider for conveying data or a file between processes during drag-and-drop or copy-and-paste activities, or from a host app to an app extension.

iOS 8.0+ | iPadOS 8.0+ | Mac Catalyst 13.1+ | macOS 10.10+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class NSItemProvider
```

Overview

Starting in iOS 11, item providers play a central role in drag and drop, and in copy and paste. They continue to play a role with app extensions.

The system uses an internal queue when calling the completion blocks for the `NSItemProvider` class. When using an item provider with drag and drop, ensure that UI updates take place on the main queue as follows:

```
DispatchQueue.main.async {
    // Work that impacts the user interface.
}
```

App extension support

An app extension typically encounters item providers when examining the `attachments` property of an `NSExtensionItem` object. During that examination, the extension can use the `hasItemConformingToTypeIdentifier(_:)` method to look for data that it recognizes. Item providers use Uniform Type Identifiers values to identify the data they contain. After finding a type of data that your extension can use, it calls the `loadItem(forTypeIdentifier:options:`

`completionHandler:)` method to load the actual data, which is delivered to the provided completion handler.

You can create item providers to vend data to another process. An extension that modifies an original data item can create a new `NSItemProvider` object to send back to the host app. When creating data items, you specify your data object and the type of that object. You can optionally use the `previewImageHandler` property to generate a preview image for your data.

A single item provider may use custom blocks to provide its data in many different formats. When configuring an item provider, use the `registerItem(forTypeIdentifier:loadHandler:)` method to register your blocks and the formats each one supports. When a client requests data in a particular format, the item provider executes the corresponding block, which is then responsible for coercing the data to the appropriate type and returning it to the client.

Topics

Creating an item provider

`convenience init?(contentsOf: URL!)`

Provides data-backed content from an existing file.

`convenience init(contentsOf: URL, contentType: UTType?, openInPlace: Bool, coordinated: Bool, visibility: NSItemProviderRepresentationVisibility)`

Provides data-backed content from an existing file with the specified parameters.

`init(item: (any NSSecureCoding)?, typeIdentifier: String?)`

Creates an item provider with an object, according to the item provider type coercion policy.

`init()`

Creates an empty item provider to which you can later register a data or file representation.

`convenience init(object: any NSItemProviderWriting)`

Creates a new item provider, employing a specified object's type identifiers to specify the data representations eligible for the provider to load.

Configuring the provider

`var preferredPresentationSize: CGSize`

The ideal presentation size of the item.

`var preferredPresentationStyle: NSItemProvider.PreferredPresentationStyle`

The preferred style for presenting the item provider's data.

`enum PreferredPresentationStyle`

The presentation styles that determine how a view shows an item provider's data.

`var suggestedName: String?`

The filename to use when writing the provided data to a file on disk.

`var teamData: Data?`

The collection of data an app uses to hold private team information during drag and drop.

Querying the provider's contents

`func canLoadObject(ofClass: any NSItemProviderReading.Type) -> Bool`

Returns a Boolean value indicating whether an item provider can load objects of a specified class.

`func canLoadObject<T>(ofClass: T.Type) -> Bool`

Returns a Boolean value indicating whether an item provider can load objects of a specified class.

`func hasItemConformingToTypeIdentifier(String) -> Bool`

Returns a Boolean value indicating whether an item provider contains a data representation conforming to a specified universal type identifier file options parameter with a value of zero.

`func hasRepresentationConforming(toTypeIdentifier: String, fileOptions: NSItemProviderFileOptions) -> Bool`

Returns a Boolean value indicating whether an item provider contains a data representation conforming to a specified universal type identifier and to specified open-in-place behavior.

`var registeredTypeIdentifiers: [String]`

Returns the array of type identifiers for the item provider, in the same order they were registered.

`func registeredTypeIdentifiers(fileOptions: NSItemProviderFileOptions) -> [String]`

Returns an array with a subset of type identifiers for the item provider, according to the specified file options, in the same order they were registered.

Loading the provider's contents

```
func loadItem(forTypeIdentifier: String, options: [AnyHashable : Any]?, completionHandler: NSItemProvider.CompletionHandler?)
```

Loads the item's data and coerces it to the specified type.

```
func loadDataRepresentation(forTypeIdentifier: String, completion Handler: (Data?, (any Error)?) -> Void) -> Progress
```

Asynchronously copies the provided, typed data into a generic data object, returning a progress object.

```
func loadDataRepresentation(for: UTType, completionHandler: (Data?, (any Error)?) -> Void) -> Progress
```

Asynchronously copies the universal type data into a generic data object, returning a progress object.

```
func loadFileRepresentation(forTypeIdentifier: String, completion Handler: (URL?, (any Error)?) -> Void) -> Progress
```

Asynchronously writes a copy of the provided, typed data to a temporary file, returning a progress object.

```
func loadFileRepresentation(for: UTType, openInPlace: Bool, completion Handler: (URL?, Bool, (any Error)?) -> Void) -> Progress
```

Asynchronously writes a copy of the universal type data to a temporary file, returning a progress object.

```
func loadInPlaceFileRepresentation(forTypeIdentifier: String, completionHandler: (URL?, Bool, (any Error)?) -> Void) -> Progress
```

Asynchronously opens a file in place, if possible, returning a progress object.

```
func loadObject(ofClass: any NSItemProviderReading.Type, completion Handler: ((any NSItemProviderReading)?, (any Error)?) -> Void) -> Progress
```

Asynchronously loads an object of a specified class to an item provider, returning a progress object.

```
func loadObject<T>(ofClass: T.Type, completionHandler: (T?, (any Error )?) -> Void) -> Progress
```

Asynchronously loads an object of a specified class to an item provider, returning a progress object.

```
func loadTransferable<T>(type: T.Type, completionHandler: (Result<T, any Error>) -> Void) -> Progress
```

Asynchronously loads an object of a specified transferable type to an item provider, returning a progress object.

Loading a preview image

```
func loadPreviewImage(options: [AnyHashable : Any]!, completionHandler: NSItemProvider.CompletionHandler!)
```

Loads the preview image for the item that the item provider represents.

```
var previewImageHandler: NSItemProvider.LoadHandler?
```

The custom preview image handler block for the item provider.

Registering CloudKit shares

```
func registerCloudKitShare(CKShare, container: CKContainer)
```

Registers a CloudKit share for the user to modify.

```
func registerCloudKitShare(preparationHandler: ((CKShare?, CKContainer?, (any Error)?) -> Void) -> Void)
```

Registers a handler that prepares a new CloudKit share.

```
func registerCKShare(CKShare, container: CKContainer, allowedSharingOptions: CKAllowedSharingOptions)
```

Registers an existing collaboration object on a server.

```
func registerCKShare(container: CKContainer, allowedSharingOptions: CKAllowedSharingOptions, preparationHandler: () async throws -> CKShare)
```

Creates and registers a new collaboration object using a collection of records to share.

Registering content types

```
var registeredContentTypes: [UTType]
```

Registered content types in the order the app registers each type.

```
var registeredContentTypesForOpenInPlace: [UTType]
```

Registered content types that the system can load as open-in-place files.

```
func registeredContentTypes(conformingTo: UTType) -> [UTType]
```

Returns an array of registered content types that conform to a specified content type.

Registering data

```
func registerDataRepresentation(forTypeIdentifier: String, visibility: NSItemProviderRepresentationVisibility, loadHandler: ((Data?, (any Error)?) -> Void) -> Progress?)
```

Registers a data-backed representation for an item, specifiying item visibility and a load handler.

```
func registerDataRepresentation(for: UTType, visibility: NSItemProviderRepresentationVisibility, loadHandler: ((Data?, (any Error)?) -> Void) -> Progress?)
```

Registers a data-backed representation for an item, specifiying item visibility and a load handler.

```
func registerItem(forTypeIdentifier: String, loadHandler: NSItemProvider.LoadHandler)
```

Lazily registers an item, according to the item provider type coercion policy.

Registering files

```
func registerFileRepresentation(forTypeIdentifier: String, fileOptions: NSItemProviderFileOptions, visibility: NSItemProviderRepresentationVisibility, loadHandler: ((URL?, Bool, (any Error)?) -> Void) -> Progress?)
```

Registers a file-backed representation for an item, specifying file options, item visibility, and a load handler.

```
func registerFileRepresentation(for: UTType, visibility: NSItemProviderRepresentationVisibility, openInPlace: Bool, loadHandler: ((URL?, Bool, (any Error)?) -> Void) -> Progress?)
```

Registers a file-backed representation for an item with item visibility, an open-in-place option, and a load handler.

Registering group activities

```
func registerGroupActivity<ActivityType>(ActivityType)
```

Registers a group activity instance with the specificed options.

```
func registerGroupActivity<ActivityType>(preparationHandler: () async throws -> ActivityType)
```

Registers a group activity instance asynchronously with the specified options.

Registering objects

```
func registerObject(any NSItemProviderWriting, visibility: NSItemProviderRepresentationVisibility)
```

Adds representations of a specified object to an item provider, based on the object's implementation of the item provider writing protocol, and adhering to a visibility specification.

```
func registerObject(ofClass: any NSItemProviderWriting.Type, visibility: NSItemProviderRepresentationVisibility, loadHandler: (((any NSItemProviderWriting)?, (any Error)?) -> Void) -> Progress?)
```

Lazily adds representations of a specified object class to an item provider, based on the object's implementation of the item provider writing protocol, and adhering to a visibility specification.

```
func registerObject<T>(ofClass: T.Type, visibility: NSItemProviderRepresentationVisibility, loadHandler: ((T?, (any Error)?) -> Void) -> Progress?)
```

Lazily adds representations of a specified object type to an item provider, based on the object's implementation of the item provider writing protocol, and adhering to a visibility specification.

```
func register<T>(@autoclosure () -> T)
```

Adds representations of a specified transferable type to an item provider.

Getting the provider's frame

```
var sourceFrame: NSRect
```

The rectangle that the item occupies in the host app's source window.

```
var containerFrame: NSRect
```

The rectangle of the item's visible content.

Constants

```
typealias CompletionHandler
```

A block that receives the item provider's data.

`typealias` `LoadHandler`

A block that loads the item provider's data and coerces it to the specified type.

Options Dictionary Key

Keys indicating options to use when generating the item provider's data.

Keys for Items Accessed in JavaScript Code

Keys in property list items that the system recieves from or sends to JavaScript code.

`class let` `errorDomain: String`

The error domain associated with the item provider.

`struct` `NSItemProviderFileOptions`

Data-access specifications that declare how to handle items.

`protocol` `NSItemProviderReading`

The protocol for implementing a class to allow an item provider to create an instance of the class.

`protocol` `NSItemProviderWriting`

The protocol for implementing a class to allow an item provider to retrieve data from an instance of the class.

`enum` `NSItemProviderRepresentationVisibility`

Specifications that control which categories of processes can see an item.

`enum` `ErrorCode`

The error codes that describe problems with consuming data from an item provider.

Relationships

Inherits From

`NSObject`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`

CustomStringConvertible
Equatable
Hashable
NSCopying
NSObjectProtocol

See Also

Attachments

`class` NSExtensionItem

An immutable collection of values representing different aspects of an item for an extension to act upon.

{} Add Functionality to Finder with Action Extensions

Implement Action Extensions to provide quick access to commonly used features of your app.