Sample Code

Building a document browser app for custom file formats

Implement a custom document file format to manage user interactions with files on different cloud storage providers.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 12.0+

Overview

Users can store their documents on cloud storage providers such as iCloud Drive. An app based on the document browser view controller allows users to browse and access their documents no matter where those documents are stored. In addition, the document browser view controller lets users create new documents, and acts as a springboard into your application's main user interface.

This sample app illustrates how to use the document browser view controller. It registers a custom file format called Particles in the system, and allows the user to create new Particles documents on any of the user's activated file providers. When the user chooses a document, the app presents an editor view, where the user can modify the document's contents. When the user completes their modifications, the system saves the file, and the user returns to the document browser view controller.

The app also demonstrates proper file handling using the `UIDocument` class, and builds a Quick Look preview and thumbnail extension for the Particles file format. Finally, this sample illustrates how to customize the appearance of the document browser view controller, the custom browser actions, and the presentation of document view controllers with a custom zoom transition.

The `UIDocumentBrowserViewController` and this Particles app require at least iOS 13.2.

Configure the sample code project

Xcode provides a Document Based App template for creating document-based apps. This template comes with everything you need to implement a document-based application from scratch.

Once you have selected the template, the project provides a storyboard, which uses a `UIDocumentBrowserViewController` as its entry point. Additionally, the template creates a `UIDocument` subclass, Document, which acts as the representation of a document of your application at runtime. The logic to create new documents is already in place. When the user creates a new document, or opens an existing one by tapping a file in the document browser view controller, the system presents a `DocumentViewController`, which has a reference to a `Document` instance. This `DocumentViewController` acts as the main user interface to view and modify the contents of a document.

> Note
>
> If you want to migrate an existing document-based application to use the `UIDocumentBrowserViewController`, make sure to present the document browser view controller full-screen as the first user interface the user sees when launching your application. Make sure to implement the `UIDocumentBrowserViewControllerDelegate` protocol and assign an instance of it to the document browser view controller.

Next, you can extend the project to fit your needs. Set up a custom file format in the exported Uniform Type Identifiers (UTIs) of your application, or configure one or more existing UTIs in the imported UTIs section. The application's document types need to be configured as well, so that the document browser view controller can display the correct files to the user.

Next, provide the document to start with when the user creates a new document; for instance, by copying a file from the application bundle, which represents a blank version of a new document. Hand over the blank file to the document browser view controller via the import handler. In order to provide proper document saving and loading, augment the `UIDocument` subclass with the logic needed in order to encode and decode the data of a document properly.

Finally, you can configure the document browser view controller to show custom browser actions to the user when selecting one or more documents at once, or when long-pressing a document to reveal the menu.

Animate view controller transitions

When presenting a document view controller after creating a new document or choosing an existing one, a transition controller can animate the view controller change with an elegant zoom transition.

The `UIDocumentBrowserViewController` class provides a method to obtain a transition controller. The transition controller has a reference to a document, so that its thumbnail can be used as the starting point or end point of the animation.

Assign a transitioning delegate object to the document view controller (the view controller that displays the document's contents). The transitioning delegate needs to conform to the `UIViewControllerTransitioningDelegate` protocol. The system asks the delegate for an animation controller when the system is about to present or dismiss the view controller. The delegate returns a transition controller obtained from the document browser view controller. The transition controller displays the zoom animation when opening or closing documents.

Configure a custom file format

This sample introduces a custom file format called Particles. Particles files use the `.particles` extension. In order to register the file format, the sample has modifies both the Exported UTIs and the Document Types in the target's Info pane.

- In the Exported UTIs section, the sample project defines a Particles document as conforming to `public.data, public.content`. This entry registers the file format with the system.
- Under "Additional exported UTI properties," the sample project defines the `UTTypeTagSpecification` dictionary with a `public.filename–extension` key and value of `particles` This entry sets `particles` as the document's extension.
- In the Document Types section, the sample project also defines the Particles document, enabling support for Particle documents in the app's document browser.
- Because this app created the file format, under "Additional document type properties," it sets the `LSHandlerRank` to `Owner`.

Preview custom documents

The sample provides two Quick Look extensions:

- The `ParticlesPreview` extension, which generates preview views.
- The `ParticlesThumbnails` extension, which generates thumbnails for files of the newly registered file format.

In order for Quick Look to choose the extensions when dealing with Particles documents, the `Info.plist` file of both extensions are configured with the Particles file format data.

Apply best practices

Whenever possible, create document-based apps using `UIDocumentBrowserViewController` and `UIDocument` subclasses. The `UIDocument` class provides the commonly expected features of a document-based application, such as saving and loading documents, asynchronous reading and writing of data, versioning with conflict detection, and much more. Additionally, `UIDocument` provides the automatic coordinated reading and writing needed to avoid problems when accessing a file on disk that could be read or written by other processes at the same time. To access a file manually, use file coordination in order to avoid data loss.

Avoid listing high-level UTIs in the document types configured for an application. Only list the UTIs that the application can actually handle. Otherwise, the document browser view controller may display files of unsupported file formats that are irrelevant to the user in the Recents section, in search results, or other locations.

Configure the document types and the exported and imported UTIs in the application's `Info.plist` file correctly. Dynamic sections such as Recent Documents, collections of tagged documents, and the popover of the application need this information to work properly.

Finally, it's important to know when to use the picker view controller. The `UIDocumentPickerViewController` and `UIDocumentBrowserViewController` are two different view controllers, each with their own purpose. Use the `UIDocumentPickerViewController` to allow the user to quickly pick an existing file to, for example, insert into the currently opened document, or to export a document to a certain location. Use the `UIDocumentBrowserViewController` as the entry point in the application to let the users create new or choose an existing document.

See Also

Documents and directories

Customizing a document-based app's launch experience

Add unique elements to your app's document launch scene.

Adding a document browser to your app

Give users access to their local or remote documents from within your app.

Providing access to directories

Use a document picker to access the content of a directory outside your app's container.

Building a document browser-based app

Use a document browser to provide access to the user's text files.

`class UIDocumentViewController`

A view controller that manages and presents a document stored locally or in the cloud.

`class UIDocumentBrowserViewController`

A view controller for browsing and performing actions on documents that you store locally and in the cloud.

`class UIDocumentPickerViewController`

A view controller that provides access to documents or destinations outside your app’s sandbox.

`class UIDocumentInteractionController`

A view controller that previews, opens, or prints files with a file format that your app can’t handle directly.