Sample Code

Building a document browser-based app

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

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 13.0+

Overview

The Document Browser sample code uses a `UIDocumentBrowserViewController` as the app's root view controller. The browser defines the structure of the app, and the app displays the browser view when it launches. Users can then use the browser to:

- Browse all the text files on the user's device, in their iCloud drive, or in any supported third-party file providers.
- Create new text files.
- Open text files.

When the user opens a file, the app transitions to an editor view. There, the user can edit and save the text file. When they are done editing, the app returns to the browser, letting the user open or create another file.

This sample code project demonstrates all the required steps to set up the document browser, to work with the user's files, and to enable system animations. The following sections describe these steps in more detail.

Setting up the document browser

The Document Browser app performs the following setup and configuration steps:

1. Assigns a `UIDocumentBrowserViewController` subclass as the window's `rootViewController`.
2. Specifies the supported document types.
3. Customizes the document browser's behavior.

Document browser-based apps must assign a `UIDocumentBrowserViewController` instance as the app's `rootViewController`, ensuring that the browser remains in memory throughout the app's lifetime.

The sample code defines a `UIDocumentBrowserViewController` subclass named `DocumentBrowserViewController`. It then marks the subclass as the app's initial view controller in the `Main.storyboard` storyboard, and displays the browser view when launched.

All document browser-based apps must also declare the document types that they can open. The sample code app declares support for text files in the project editor's Info pane. For more information on setting the document type, see Setting up a document browser app.

Finally, the sample code configures the document browser in the `DocumentBrowserViewController` class's `viewDidLoad()` method. Specifically, it enables document creation, and disables multiple document selection. This lets users create new documents from the browser, while also preventing them from opening more than one document at a time.

```
allowsDocumentCreation = true
allowsPickingMultipleItems = false
```

For more information on configuring a document browser, see Customizing the browser.

Creating new documents

When the user creates a new document, the system calls the document browser delegate's `documentBrowser(_:didRequestDocumentCreationWithHandler:)` method.

```
// Create a new document.
func documentBrowser(_ controller: UIDocumentBrowserViewController,
                     didRequestDocumentCreationWithHandler importHandler: @escaping

    os_log("==> Creating A New Document.", log: .default, type: .debug)

    let doc = TextDocument()
    let url = doc.fileURL

    // Create a new document in a temporary location.
```

```
    doc.save(to: url, for: .forCreating) { (saveSuccess) in

        // Make sure the document saved successfully.
        guard saveSuccess else {
            os_log("*** Unable to create a new document. ***", log: .default, type:

            // Cancel document creation.
            importHandler(nil, .none)
            return
        }

        // Close the document.
        doc.close(completionHandler: { (closeSuccess) in

            // Make sure the document closed successfully.
            guard closeSuccess else {
                os_log("*** Unable to create a new document. ***", log: .default, ty

                // Cancel document creation.
                importHandler(nil, .none)
                return
            }

            // Pass the document's temporary URL to the import handler.
            importHandler(url, .move)
        })
    }
}
```

In this method, the app creates, saves, and then closes a new text document. If successful, the app passes the URL to the method's import handler, requesting that the system move the document to its final location. Otherwise, it passes `nil` to the import handler, canceling document creation.

Opening and importing documents

Users can open documents in multiple ways. The sample code handles the following situations:

- If the app imports a document (including successfully creating a new document), the system calls the `documentBrowser(_:didImportDocumentAt:toDestinationURL:)` method.
- If the user selects a document from the browser, the system calls the `documentBrowser(_:didPickDocumentURLs:)` method.

- If the user shares a document with the app, or drags a document into the app, the system calls the app delegate's `application(_:open:options:)` method.

In the first two cases, the app calls the custom `presentDocuments(at:)` method. In the third case, the app calls the browser's `revealDocument(at:importIfNeeded:completion:)` method to import the document (if needed). It then calls the `presentDocuments(at:)` method.

The `presentDocuments(at:)` method instantiates a `TextDocumentViewController`, sets up the document's animation, opens the document, and then presents the view controller by calling the browser's `present(_:animated:completion:)` method.

Enabling animations

The document browser provides two built-in animations: one for loading a file, another for transitioning to and from the document view.

To enable either of the system-provided document browser animations, first you need to request a transition controller for the document by calling the `transitionController(forDocumentURL:)` method.

```
transitionController = transitionController(forDocumentAt: documentURL)
```

To enable the loading animation, assign a `Progress` object to the transition controller when you begin to load the document.

```
// Set up the loading animation.
transitionController!.loadingProgress = doc.loadProgress
```

Increment the progress as the document loads, making sure to mark it as complete as soon as loading finishes. To simulate slow, incremental loading in the sample code project, uncomment the `TextDocument` class's `read(from:)` method.

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 app for custom file formats

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

`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.