< How to delete Core Data objects from SwiftUI views

How to export files using ExportFilesAction >

How to create a document-based app using FileDocument and DocumentGroup

Paul Hudson @twostraws June 23rd 2020

Updated for Xcode 12.0

New in iOS 14

SwiftUI comes with support for document-based apps, which are apps that let users create, edit, and share documents such as text files. In SwiftUI we're given two main types to work with: the `FileDocument` protocol to define what a document in our app looks like, and the `DocumentGroup` struct that gives us a default scene to let users create, open, and save documents.

Creating your own document-based app take three steps:

1. Defining what your document is, including how it should be saved and loaded.
2. Creating some sort of view that lets users edit their documents.
3. Creating a `DocumentGroup` capable of creating your files and loading them into your user interface.

We'll work through each of those here, starting with defining what your document is. Some document types save multiple files of different types, but for now we're going to say that we have support only plain text, and we want that text to be read and written directly to disk.

First, add `import UniformTypeIdentifiers` to the top of your Swift file, so you can bring in uniform type identifiers – a fixed way of saying what data types your document can work with.

Now add this struct, defining a simple text file:

```
struct TextFile: FileDocument {
    // tell the system we support only plain text
    static var readableContentTypes = [UTType.plainText]

    // by default our document is empty
    var text = ""

    // a simple initializer that creates new, empty documents
    init(initialText: String = "") {
        text = initialText
    }

    // this initializer loads data that has been saved previously
    init(fileWrapper: FileWrapper, contentType: UTType) throws {
        if let data = fileWrapper.regularFileContents {
            text = String(decoding: data, as: UTF8.self)
        }
    }

    // this will be called when the system wants to write our data to disk
    func write(to fileWrapper: inout FileWrapper, contentType: UTType) throw
        let data = Data(text.utf8)
        fileWrapper = FileWrapper(regularFileWithContents: data)
    }
}
```

Notice how in the `write(to:)` method we convert our text string into a `Data` instance, then save that using a `FileWrapper`. It's not our job to say where the file should be stored – iOS takes care of that for us.

Our second task is to create some sort of editor area where the user can edit our text. This should use an `@Binding` property wrapper so that it updates the text in our `TextFile` struct rather than keeping its own local copy:

```
struct ContentView: View {
    @Binding var document: TextFile

    var body: some View {
        TextEditor(text: $document.text)
    }
}
```

And now our final step is to edit the main Swift file for the project to include a `DocumentGroup`, which presents the system-standard interface for browsing, opening, and creating files:

```
@main
struct YourAwesomeApp: App {
    var body: some Scene {
        DocumentGroup(newDocument: TextFile()) { file in
            ContentView(document: file.$document)
        }
    }
}
```

That's it! Your document-based app is ready to go.

Similar solutions...

- SwiftUI tips and tricks
- How to use Instruments to profile your SwiftUI code and identify slow layouts
- Building a menu using List
- Two-way bindings in SwiftUI
- How to position views in a grid using LazyVGrid and LazyHGrid

< How to delete Core Data objects from SwiftUI views

How to export files using ExportFilesAction >

Was this page useful? Let us know!

★★★★★

Average rating: 3.7/5

@twostraws

paul@hackingwithswift.com

Sponsor the site