Article

Sharing data between your App Clip and your full app

Use CloudKit, Sign in with Apple, shared user defaults or containers, and the keychain to offer a smooth transition from your App Clip to your app.

Overview

When users of your App Clip install your full app, the full app replaces the App Clip and receives all invocations in its place. It's important you provide a good user experience to users who have become familiar with the App Clip and are now starting to use the full app; for example:

- Show information in the app that the user entered when they used the App Clip.
- Make downloaded data available to the app.
- Don't require users to log in to their account again.

Making App Clip data accessible to your full app is key to offering a smooth transition from App Clip to full app. Choose from the following technologies:

- Access your public iCloud database in your App Clip.
- Store App Clip data and assets in a shared container and access them in your full app.
- Store sensitive App Clip data in the keychain and access it in your full app.
- Use Sign in with Apple to offer a seamless login experience.

Access your public iCloud database

Making sure your App Clip has access to the same data as your full app and keeping its data up to date are important things to consider when you create your App Clip. For example, an App Clip for restaurants needs to always show the latest weekly lunch menu that's also available to users of your full app. Starting with iOS 16, your App Clip can use CloudKit to read data you store in a public iCloud database with CloudKit — including a Core Data store you mirror with a public CloudKit database. This enables your App Clip to access information that's available to all users of your app — like the previously mentioned lunch menu for a restaurant app and App Clip.

To read your app's public iCloud database, add the iCloud capability to your App Clip as described in Configuring iCloud services. The process is the same as for your full app. However, your App Clip can only set the `CloudKit-Anonymous` value for the `iCloud Services Entitlement`. Xcode chooses this value for you when you add the iCloud capability to your App Clip target. After you've added the iCloud capability to your App Clip target, specify the container your App Clip should access, and access your data just like you do in your regular app.

Important

Your App Clip can't write data to the public database. It can't use iCloud Documents, iCloud key-value storage, or private and shared containers.

For more information on iCloud and public databases, refer to Build Apps Using CloudKit and Designing apps using CloudKit.

Make local App Clip data available to the full app

Your App Clip can make its local data accessible to your corresponding full app by storing data in a shared container. The full app can then access this local data when it replaces the App Clip, providing a positive user experience.

To store local data in a shared container:

1. Add the App Groups capability to the targets of both the App Clip and the full app. For more information about configuring App Groups, refer to Configuring app groups.
2. For both targets, add the same app group to the capability; for example, `group.exampleApp.appClipMigration`.
3. Add code to your App Clip target that obtains the URL of the shared container using `containerURL(forSecurityApplicationGroupIdentifier:)` and store data using this URL; for example, by using `write(to:atomically:encoding:)`.
4. In your full app's code, use the same function to obtain the URL of the shared container and access its content; for example, by using `init(contentsOfURL:encoding:)`.

Tip

App Clips support Background Assets. Make sure to add support for background assets to your app and your App Clip. Background Assets use app groups, and assets your App Clip downloads become available to your full app.

In addition to a shared container, the App Clip can also store information in a shared `UserDefaults` instance that's accessible to the full app. The following code uses the configured app group to create the shared `UserDefaults` instance and store a string:

```
guard let sharedUserDefaults = UserDefaults(suiteName: "group.exampleApp.appClipMigr
    // Error handling
}
sharedUserDefaults.set("A sample string", forKey: "sharedText")
```

When users install the full app, it can access the shared user defaults. For example, access the string stored in the previous code using the following code snippet:

```
guard let sharedUserDefaults = UserDefaults(suiteName: "group.exampleApp.appClipData
    // Error handling
}
guard let migratedData = sharedUserDefaults.string(forKey: "sharedText") else { retu
```

To preserve user privacy across apps and App Clips, an App Clip can only share its data with its corresponding app. Don't store sensitive user information, such as passwords, in a shared app container or in user defaults. However, if your App Clip requires people to log in to an account to offer its functionality, you likely want to offer a seamless upgrade experience to the full app. In this case, you might store an API key, hashed user ID, or similarly anonymized identifier in the shared app container or in user defaults. Then, you can use the identifier when the user launches the full app to offer a more seamless upgrade experience that doesn't require them to again log in.

Review keychain usage

The keychain offers a secure way to store sensitive information. Storing and reading App Clip data in the keychain generally works the same as for your full app. However, be aware of the following behavior when using the keychain:

- Your App Clip can't access data your full app stores in the keychain. For example, say your app stores login information in the keychain. A user later uninstalls the app only to later launch your

App Clip from an invocation. In the App Clip, they'll need to log in again because your App Clip can't access the stored login information that may have remained in the keychain.

- On devices that run iOS 15.3 or earlier, you can't make information the App Clip stores in the keychain accessible to its corresponding full app. If your App Clip requires people to log in to an account to offer its functionality, don't make the user sign in a second time in the full app. Instead, store an API key or hashed user ID in the shared app container as described above, or offer Sign in with Apple.

- Starting with iOS 15.4, information that the App Clip stores in the keychain is accessible to its corresponding full app. To make sure only the corresponding full app receives access to keychain items stored by the App Clip, the system uses the `com.apple.developer.associated-appclip-app-identifiers` and `Parent Application Identifiers Entitlement` entitlements. When you create an App Clip with Xcode, it adds the `Parent Application Identifiers Entitlement`. It automatically adds the `com.apple.developer.associated-appclip-app-identifiers` entitlement when you archive the app that contains the App Clip.

A user may frequently install and uninstall your app and use the App Clip before reinstalling your app. If your full app and your App Clip write the same data to the keychain, make sure your code handles the following cases:

- The keychain may contain multiple copies of the same data and they're accessible to the full app, especially if you use `kSecClassKey` because the entries have different public key hash values.

- A keychain item by the App Clip might overwrite an item that a previous installation of the full app created.

These cases are especially common when a person frequently installs and uninstalls the full app and launches the App Clip in between — for example, when you develop your app and your App Clip at the same time. To avoid errors caused by duplicated or overwritten keychain data, add code that:

- Checks if more than one copy of an item exists in the keychain

- Uses different values for `kSecAttrLabel` to distinguish between items stored by your app and your App Clip

- Decides which information to use if there's a duplicate

- Deletes any unneeded items stored in the keychain

The following example shows how your App Clip can store a sensitive string in the keychain. Note how the code sets the `kSecAttrLabel` attribute to the `fruta-appclip` value to distinguish it from a keychain entry that the app creates.

```swift
// Write sensitive information you use in your App Clip to the keychain — for exampl
let addSecretsQuery: [String: Any] = [
    kSecClass as String: kSecClassGenericPassword,
    kSecValueData as String: "smoothie-secret".data(using: .utf8),
    kSecAttrLabel as String: "fruta-appclip"
]
SecItemAdd(addSecretsQuery as CFDictionary, nil)
```

When a user installs the full app, read the sensitive information that the App Clip stored as shown in the following example:

```swift
// Read the sensitive information from the keychain that your App Clip stored.
var readSecretsQuery: [String: Any] = [
    kSecClass as String: kSecClassGenericPassword,
    kSecReturnAttributes as String: true,
    kSecAttrLabel as String: "fruta-appclip",
    kSecReturnData as String: true
]
var secretsCopy: AnyObject?
SecItemCopyMatching(readSecretsQuery as CFDictionary, &secretsCopy)
```

For more information on using the keychain to store sensitive information, refer to Using the keychain to manage user secrets and Keychain services.

Offer Sign in with Apple

App Clips may use any technology to allow users to sign in to their account or create one. However, if your App Clip requires users to log in to an account, consider offering Sign in with Apple. In addition to helping provide a simple, secure, and privacy-preserving account creation and sign-in experience, it offers a positive experience when a user starts using the full app.

If you offer Sign in with Apple and create a shared data container, create a user experience that doesn't require users to log in again when they start using the full app. The following code stores the `ASAuthorizationAppleIDCredential` in a shared `UserDefaults` instance:

```swift
let groupUserDefaults = UserDefaults(suiteName: "group.com.example.appClipDataMigrat
guard let credential = authorization.credential as ASAuthorizationAppleIDCredential
groupUserDefaults?.set(credential.user, forKey: "SavedUserID")
```

In the app’s code, retrieve the stored Apple ID authorization credential and use it to sign in the user:

```
let provider = ASAuthorizationAppleIDProvider()
let groupUserDefaults = UserDefaults(suiteName: "group.com.example.appClipDataMigrat
let user = groupUserDefaults?.get("SavedUserID")
provider.getCredentialState(forUserID: user) { state, error in
    if state == .authorized {
        readFavoriteSmoothies(user)
    }
}
```

See Also

App Clip to full app transition

Recommending your app to App Clip users

Display an overlay in your App Clip to recommend your app to users.