Sample Code

Synchronizing App Preferences with iCloud

Store app preferences in iCloud and share them among instances of your app running on a user's connected devices.

Download

iOS 13.5+ | iPadOS 13.5+ | macOS 10.13+ | Xcode 11.6+

Overview

This sample demonstrates how to store preference, configuration, and app-state data in the iCloud key-value store using the `NSUbiquitousKeyValueStore` class. You can then share this data with every instance of your app on every device connected to a user's iCloud account.

In this sample the user picks a background color. App instances running on other devices that are logged in with the same iCloud account are then notified of the color to match.

> **Note**
>
> With `NSUbiquitousKeyValueStore` you're limited to 1 MB of total space. Don't use it to store large amounts of data.

The key-value store is not a replacement for `UserDefaults` or other local techniques for saving the same data. The key-value store's purpose is to synchronize key-value data between app instances running on different devices of the same user. If iCloud is not enabled or is not available on a given device, the app still keeps a local copy of the data in `NSUserDefaults`.

Configure the Sample Code Project

To configure your Xcode project first you need change the Bundle Identifiers for both macOS and iOS targets to match your own. Next, to work with iCloud and `NSUbiquitousKeyValueStore`, you need set up an iCloud key-value store ID for both targets. This sample already has the both targets' iCloud capabilities set up, with the `PrefsInCloud.entitlements` file already created. You need to fill in your own key-value store IDs in those entitlement files to complete the configurations.

For each target (iOS and macOS):

1. In the Xcode project, open the target's `PrefsInCloud.entitlements` file.
2. In that file, go to `com.apple.developer.ubiquity-kvstore-identifier` entitlement and replace `$(CFBundleIdenfier)` with your key-value store ID (i.e. `$(TeamIdentifierPrefix)<your key-value_store ID>`). For example if your key-value store ID is "com.mycompany.myKeyValueStoreID" it would look like this: `$(TeamIdentifierPrefix)com.mycompany.myKeyValueStoreID`. It's important that the key-value store ID portions are the same between both targets.

In addition to configuring the project, you need to configure the devices in which the project runs.

To configure the iOS and macOS devices, verify that iCloud Drive is turned on. Then log into both devices with the same iCloud account.

Build and run the app to install it on both devices, and then do the following:

- iOS: Tap the action button at the top right to pick the desired background color.
- macOS: Choose the desired background color from window's popup menu.

The desired color represents a value (numbered 0 to 3) which is uploaded to iCloud. App instances running on the other devices logged into the same iCloud account are notified to match that specific background color.

> **Important**
>
> The key-value store is intended for storing data that changes infrequently. As you test your devices, if the app on a device makes frequent changes to the key-value store, the system may defer the synchronization of some changes in order to minimize the number of round trips to the server. The more frequently the app make changes, the more likely the changes will be deferred and will not immediately show up on the other devices.

Register to Respond to Key-Value Store Changes

The sample uses `NotificationCenter` to register as an observer to the notification `didChangeExternallyNotification`. The following example installs the app as the observer to respond to the key-value store changes:

```
NotificationCenter.default.addObserver(self,
    selector: #selector(ubiquitousKeyValueStoreDidChange(_:)),
    name: NSUbiquitousKeyValueStore.didChangeExternallyNotification,
    object: NSUbiquitousKeyValueStore.default)
```

Immediately following the call to `addObserver`, obtain the key-value store changes since last launch by calling `synchronize()`. The operating system controls when keys and values are uploaded to iCloud. Calling `synchronize()` simply stores a local cache of them and notifies iCloud of new data waiting to be uploaded.

Respond to Key-Value Store Changes

The sample uses `NSNotificationCenter` to register as an observer to the notification `didChangeExternallyNotification` to detect for value changes in iCloud.

The sample implements the function `ubiquitousKeyValueStoreDidChange` for listening for key-value notification changes. This function is called when the key-value store in the cloud has changed externally. It replaces the old key-value color with the new one. Additionally, this function updates the `NSUserDefaults` key-value.

This sample examines the reason for the change through the use the `NSUbiquitousKeyValueStoreChangeReasonKey` from the notification's `userInfo`.

```
guard let userInfo = notification.userInfo else { return }

// Get the reason for the notification (initial download, external change or quota v
guard let reasonForChange = userInfo[NSUbiquitousKeyValueStoreChangeReasonKey] as? I
```

The reason for the key-value notification change is one of the following:

- `NSUbiquitousKeyValueStoreServerChange`: Value(s) were changed externally from other users/devices.
- `NSUbiquitousKeyValueStoreInitialSyncChange`: Initial downloads happen the first time a device is connected to an iCloud account, and when a user switches their primary iCloud account.
- `NSUbiquitousKeyValueStoreQuotaViolationChange`: The app's key-value store has exceeded its space quota on the iCloud server.
- `NSUbiquitousKeyValueStoreAccountChange`: The user has changed the primary iCloud account.

To obtain key-values that have changed, use the key `NSUbiquitousKeyValueStoreChangedKeysKey` from the notification's `userInfo`.

```
guard let keys =
          userInfo[NSUbiquitousKeyValueStoreChangedKeysKey] as? [String] else { retu
```

```
let possibleColorIndexFromiCloud =
          NSUbiquitousKeyValueStore.default.longLong(forKey: gBackgroundColorKey)
```

See Also

App Preferences

`class` **NSUbiquitousKeyValueStore**

An iCloud-based container of key-value pairs you use to share data among instances of your app running on a user's connected devices.