Article

Optimizing Your App's Data for iCloud Backup

Minimize the space and time that backups take to create by excluding purgeable and nonpurgeable data from backups.

Overview

When iCloud Backup is in an enabled state, it periodically creates a backup of the user's device, including your app's data. By storing purgeable data in specific directories and signaling to the system that backups can exclude certain nonpurgeable data, you can shorten the time it takes to create a backup and help reduce the amount of space that backup consumes.

Exclude Purgeable Data

Purgeable data is app data that the system can delete without affecting the app's execution and that the app can re-create if it needs. For example, your app might generate thumbnails of larger images for display in a collection view. If the app saves the thumbnails to disk, consider them purgeable data because, if the system deletes those files, the app can create them again from source.

The app container provides two directories for storing purgeable data:

- `/tmp`
- `/Library/Caches`

The system periodically purges these directories, so iCloud Backup excludes them by default. If your app creates purgeable data, store it in one of these directories; otherwise, iCloud Backup may include that data, needlessly increasing the physical size of the backup with no benefit to the user. Don't use these directories to exclude nonpurgeable data from iCloud Backup.

To retrieve the location of these system-provided directories, use the `FileManager` class, as the following example shows.

```
let manager = FileManager.default

// Get the URL to the app container's 'tmp' directory.
let tmpDirectoryURL = manager.temporaryDirectory

// Get the URL to the app container's 'Caches' directory.
let cachesDirectoryURL = try manager.url(for: .cachesDirectory,
                                         in: .userDomainMask,
                                         appropriateFor: nil,
                                         create: false)
```

Consider deleting purgeable data as soon as your app is done with it so that it doesn't continue to consume space on the user's device.

Mark Nonpurgeable Data as Excludable

Nonpurgeable data is data the user creates, or data the app requires to function as the user expects. Some nonpurgeable data isn't appropriate for backup.

For example, if your app downloads high-definition movies for offline viewing, exclude those files because they're typically large in size and the user can download them again on a restored device, if necessary. Conversely, if your app allows the user to import arbitrary files such as PDFs, ebooks, and digital comics, don't exclude those files because it might be difficult, even impossible, for the user to re-create them on a restored device.

If your app creates nonpurgeable data that isn't appropriate for backup, you can indicate which files and directories the system can exclude by setting their `isExcludedFromBackup` resource value to `true`, as the following example shows.

```
func excludeItem(at url: URL) throws {
    // Create the resource values for the specified URL.
    var values = URLResourceValues()
    values.isExcludedFromBackup = true

    // Apply those values to the URL.
    var url = url
    try url.setResourceValues(values)
}
```

Note

Because certain file operations can reset resource values, make sure you set an excluded file's resource values each time you save it.

The `isExcludedFromBackup` resource value exists only to provide guidance to the system about which files and directories it can exclude; it's not a mechanism to guarantee those items never appear in a backup or on a restored device.

To indicate the system can exclude a group of related files from iCloud Backup, move those files into a directory and update the directory's `isExcludedFromBackup` resource value. If you create an excludable directory inside the app container's `Library` directory, consider naming the directory with the app's bundle identifier to avoid potential conflicts with directories the system may create there in the future.

```
let manager = FileManager.default

// Retrieve the app's bundle identifier.
if let bundleIdentifier = Bundle.main.bundleIdentifier {

    // Get the URL to the app container's 'Library' directory.
    var url = try manager.url(for: .libraryDirectory,
                              in: .userDomainMask,
                              appropriateFor: nil,
                              create: false)

    // Append the bundle identifier to the retrieved URL.
    url.appendPathComponent(bundleIdentifier, isDirectory: true)

    // Use the URL to create the new directory.
    try manager.createDirectory(at: url,
                                withIntermediateDirectories: true,
                                attributes: nil)
}
```

See Also

Files and Data Persistence

File System

Create, read, write, and examine files and folders in the file system.

- Archives and Serialization

  Convert objects and values to and from property list, JSON, and other flat binary representations.

- Preferences

  Persistently store domain-scoped pieces of information for configuring your app.

- Spotlight

  Search for files and other items on the local device, and index your app’s content for searching.

- iCloud

  Manage files and key-value data that automatically synchronize among a user’s iCloud devices.