Sample Code

Selecting Photos and Videos in iOS

Improve the user experience of finding and selecting assets by using the Photos picker.

Download

iOS 15.0+ | iPadOS 15.0+ | Xcode 13.0+

Overview

This sample shows how to use the PhotoKit Photos picker, and illustrates how to filter assets according to the user's selection. After you have a selection, you use an item provider to load the asset to display in the app. The sample also explores the new configuration options in iOS 15.

> Note
>
> This sample code project is associated with WWDC21 session 10046: Improve access to Photos in your app.

Understand the Photos Picker Benefits

Both `PHPickerViewController` and `UIImagePickerController` use the out-of-process Photos picker user interface. The `PHPickerViewController` contains a powerful set of APIs that make it a great alternative to `UIImagePickerController`. `PHPickerViewController` improves stability and reliability, and includes several benefits to developers and users, such as the following:

- Deferred image loading and recovery UI
- Reliable handling of large and complex assets, like RAW and panoramic images

- User-selectable assets that aren't available for `UIImagePickerController`
- Configuration of the picker to only display Live Photos
- Availability of `PHLivePhoto` objects without library access
- Stricter validations against invalid inputs

Apps don't need to request photo library permission when using either class, so the sample app avoids requesting permission until it's necessary. A camera app, photo editing app, or library browsing app needs to use much more of PhotoKit's functionality, but an app that's only setting a basic profile photo doesn't need photo library permission. An app that only saves photos to the photo library can use the Add Photos Only permission level when requesting authorization.

Configure the Photos Picker

Before displaying the photo library, the sample creates a `PHPickerConfiguration` object using the shared photo library. Creating a configuration without a photo library provides only asset data, and doesn't include asset identifiers.

```
var configuration = PHPickerConfiguration(photoLibrary: .shared())
```

The picker displays all asset types by default. A filter configures the picker to display videos, images with live photos, or live photos only. The sample contains three buttons that reflect the available filter types.

An app creates custom filters by combining filter options. For example, the following code displays live photos and videos:

```
var newFilter = PHPickerFilter.any(of: [.livePhotos, .videos])
```

A new feature in iOS 15 is the capability to change the selection behavior. The sample sets the selection property to `ordered`, which places a numbered checkmark when selecting items, and disables swipe to select.

The preselection API allows for presenting the picker with selected photos. This gives the user the opportunity to select more photos, or to deselect preselected photos.

```
// Set the filter type according to the user’s selection.
configuration.filter = filter
// Set the mode to avoid transcoding, if possible, if your app supports arbitrary in
configuration.preferredAssetRepresentationMode = .current
// Set the selection behavior to respect the user’s selection order.
```

```
configuration.selection = .ordered
// Set the selection limit to enable multiselection.
configuration.selectionLimit = 0
// Set the preselected asset identifiers with the identifiers that the app tracks.
configuration.preselectedAssetIdentifiers = selectedAssetIdentifiers
```

Display a Picker with the Configuration

The sample creates and displays the picker using the configuration object. Displaying the photo library doesn’t need user permission because it’s running in a separate process. An app can’t take screenshots of content and can only read the assets that the user selects.

```
let picker = PHPickerViewController(configuration: configuration)
picker.delegate = self
present(picker, animated: true)
```

Apps are responsible for presenting and dismissing the photo library. The sample adopts to the photo library delegate that notifies the app when the user cancels the flow.

See Delivering an Enhanced Privacy Experience in Your Photos App to learn more about requesting authorization and limited library capabilities.

Retrieve the Selected Results

When completing a picker session, the delegate provides a list of result objects that contain an item provider that allows for loading data asynchronously. The results contain local identifiers because of initializing the configuration with a photo library.

The sample stores the selected asset identifiers to preload them when displaying the photo library again.

```
let existingSelection = self.selection
var newSelection = [String: PHPickerResult]()
for result in results {
    let identifier = result.assetIdentifier!
    newSelection[identifier] = existingSelection[identifier] ?? result
}

// Track the selection in case the user deselects it later.
selection = newSelection
selectedAssetIdentifiers = results.map(\.assetIdentifier!)
```

```
selectedAssetIdentifierIterator = selectedAssetIdentifiers.makeIterator()

if selection.isEmpty {
    displayEmptyImage()
} else {
    displayNext()
```

Fetch Selected Asset Data for Display

An item provider gives an app the ability to load assets asynchronously and on demand. Before loading an object, the sample verifies that the object is available.

An item provider also allows for progress reporting. An asset may not be available on-device if iCloud Photos and Optimize Storage aren't active. In this case, it can take some time to download a large asset. An item provider returns a `Progress` object that conveys ongoing progress to an app.

```
guard let assetIdentifier = selectedAssetIdentifierIterator?.next() else { return }
currentAssetIdentifier = assetIdentifier

let progress: Progress?
let itemProvider = selection[assetIdentifier]!.itemProvider
if itemProvider.canLoadObject(ofClass: PHLivePhoto.self) {
    progress = itemProvider.loadObject(ofClass: PHLivePhoto.self) { [weak self] live
        DispatchQueue.main.async {
            self?.handleCompletion(assetIdentifier: assetIdentifier, object: livePho
        }
    }
}
```

Load Asset Metadata

Retrieving metadata for an asset doesn't require special permission. An app can get the properties after loading an object from an item provider. However, it may not include all properties because of the transcoding process.

The following code retrieves the property list by checking that the item provider conforms to an image type identifier. After confirming, it loads the data representation of an asset, which avoids losing information from transcoding, and allows for better metadata retrieval.

```
if itemProvider.hasItemConformingToTypeIdentifier(UTType.image.identifier) {
    itemProvider.loadDataRepresentation(forTypeIdentifier: UTType.image.identifier)
        guard let data = data,
              let cgImageSource = CGImageSourceCreateWithData(data as CFData, nil),
              let properties = CGImageSourceCopyPropertiesAtIndex(cgImageSource, 0,
        print(properties)
    }
}
```

See Also

Sample code

{} Browsing and Modifying Photo Albums

Help users organize their photos into albums and browse photo collections in a grid-based layout using PhotoKit.

{} Bringing Photos picker to your SwiftUI app

Select media assets by using a Photos picker view that SwiftUI provides.

{} Implementing an inline Photos picker

Embed a system-provided, half-height Photos picker into your app’s view.

{} Creating a Slideshow Project Extension for Photos

Augment the macOS Photos app with extensions that support project creation.