Sample Code

Creating a Slideshow Project Extension for Photos

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

Download

macOS 11.0+ | Xcode 13.0+

Overview

Starting in macOS 10.13, you can create Photos project extensions. This sample app shows you how to implement a slideshow extension that transitions between photos by zooming in on a region of interest (ROI) with significant meaning in an asset, for example, faces that are relevant to a user, as opposed to faces in a crowd. The app demonstrates the computation of saliency based on an ROI's weight and quality and the process of subscribing to change notifications so your extension can respond to asset modifications.

Configure the Sample Code Project

In the extension's `Info.plist` file, choose the extension type by entering `slideshow` in the field at `NSExtension` > `NSExtensionAttributes` > `PHProjectCategory`. Add more categories to the information property list for your extension to appear in other categories in the Create menu.

Build and run the Photos Project Slideshow scheme in Xcode once to run the sample app and install the extension in the macOS Photos app. To use the extension, build and run the Slideshow Sample scheme in Xcode, which prompts you to open the macOS Photos app to use the extension.

From within the Photos app, access the Create categories by choosing File > Create or right-clicking any group of assets. Under the Slideshow category, you'll see the app extension and can create a project to run in it.

Because the project extension runs inside the Photos app, the sample emulates the grid layout of the user's photo assets. Pressing the play button in the upper-right corner of the extension starts the slideshow.

Customize the Focus Rectangle of the Zoom Transition

The sample code project contains custom `Animator` and `AssetModel` classes.

The `Animator` class handles transitions between photos in the slideshow. This sample's `Animator` asks an `AssetModel` object for a rectangle to zoom in on. Photos identifies each ROI it finds as a possible ROI, and the sample uses the bounding box of the most salient one as the preferred zoom rectangle. The code defines saliency of a `PHProjectRegionOfInterest` as the sum of its `weight` and `quality` values, and then sorts the array of the photo's regions by that value.

```
let sortedRois = assetProjectElement.regionsOfInterest.sorted { (roi1, roi2) -> Bool
    return roi1.weight + roi1.quality < roi2.weight + roi2.quality
}
return sortedRois.last?.rect
```

The `weight` of an ROI represents the pervasiveness of the ROI in the project as a whole. The `quality` score represents the quality of the ROI in the individual asset, based on factors such as sharpness, visibility, and prominence in the photo. Adding these two values is a heuristic for determining the ROI's relative importance throughout a photo project.

Respond to Asset Changes in the Project

To respond to asset changes in the Photos Library, your app extension needs to monitor change notifications. Register for change observation as soon as the project begins or resumes. In the `PHProjectExtensionController` protocol, the `beginProject(with:projectInfo:completion:)` and `resumeProject(with:completion:)` methods provide points for your extension to begin monitoring changes.

```
self.projectAssets = PHAsset.fetchAssets(in: extensionContext.project, options: nil)
extensionContext.photoLibrary.register(self)
```

When the project is complete, use the `finishProject(completionHandler:)` protocol method to unregister from change observation.

```
library.unregisterChangeObserver(self)
```

Whenever something changes in the Photos library, the system calls the photoLibraryDidChange(_:) method. When implementing this method, ask the PHChange instance for details about changes to any of the objects. When assets are added or removed, the sample project calls updatedProjectInfo(from:completion:) to get an updated PHProjectInfo instance that you use to refresh your UI.

```
func photoLibraryDidChange(_ changeInstance: PHChange) {
    guard let fetchResult = projectAssets,
        let changeDetails = changeInstance.changeDetails(for: fetchResult)
        else { return }
    projectAssets = changeDetails.fetchResultAfterChanges

    guard let projectExtensionContext = projectExtensionContext else { return }
    projectExtensionContext.updatedProjectInfo(from: projectModel?.projectInfo) { (u
        guard let projectInfo = updatedProjectInfo else { return }
        DispatchQueue.main.async {
            self.setupProjectModel(with: projectInfo, extensionContext: projectExter
        }
    }
}
```

Support Copy and Paste

If your extension handles the paste action, implement the validateMenuItem: delegate method to handle pasteboard contents.

```
func validateMenuItem(_ menuItem: NSMenuItem) -> Bool {
    var canHandlePaste = false
    if menuItem.action == #selector(paste(_:)) {
        canHandlePaste = canHandleCurrentPasteboardContent()
    }
    return canHandlePaste
}
```

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.

{} Selecting Photos and Videos in iOS

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

{} 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.