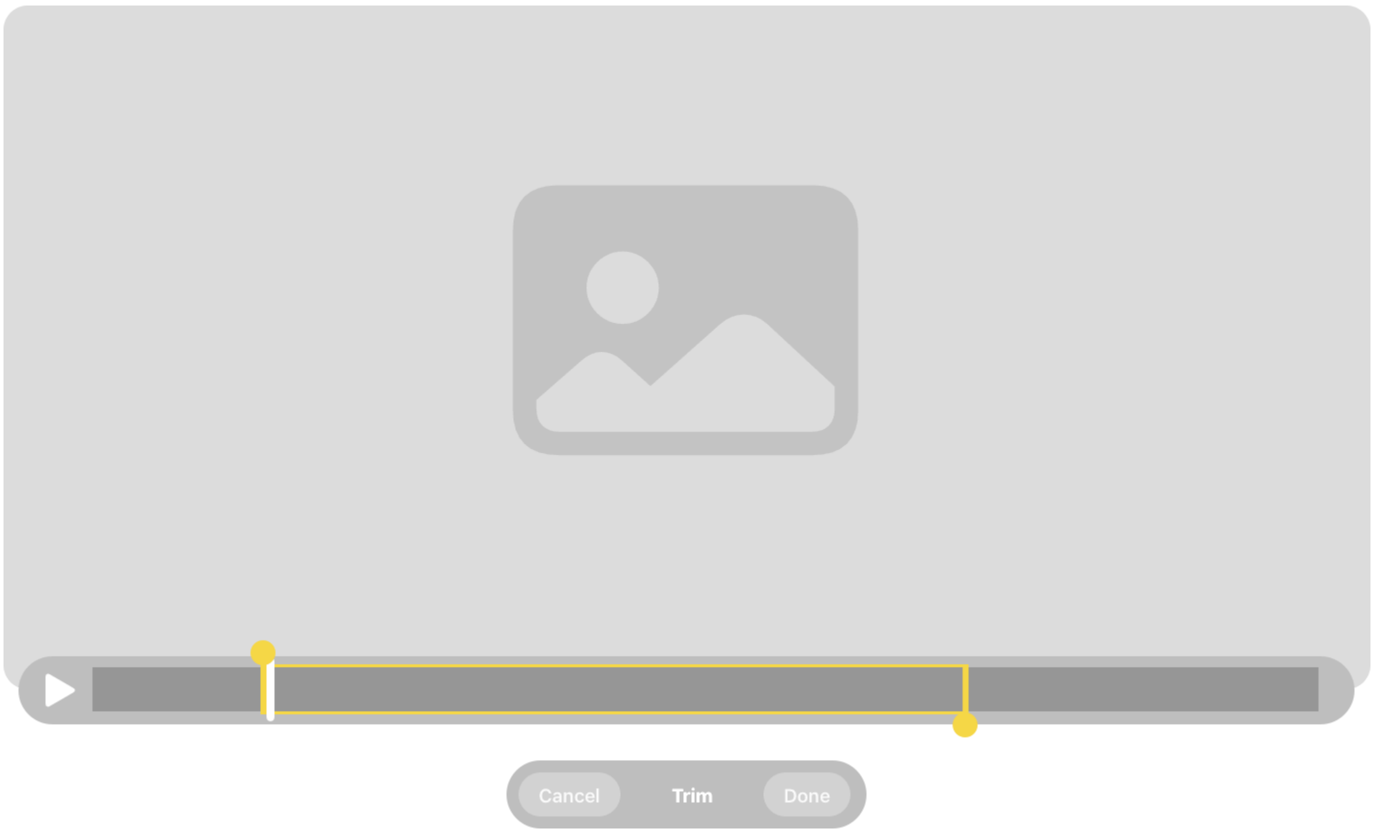
Article

Trimming and exporting media in visionOS

Display standard controls in your app to edit the timeline of the currently playing media.

Overview

You use `AVPlayerViewController` to present the system video-player interface in your visionOS app. In addition to its primary role, `AVPlayerViewController` can also provide a media-trimming experience similar to the interface of QuickTime Player in macOS, like that below.



When you enable this feature, people can specify a segment of the media timeline for display. This article describes how to adopt this feature in your app, and shows how to use AVFoundation to export the trimmed result.

Determine whether the media supports trimming

Apps typically provide a user-interface element to put the player view controller into trimming mode. Because the player doesn't support trimming certain media, such as HTTP Live Streaming or protected content, apps observe the state of the `canBeginTrimming` property to update the enabled state of their user interface accordingly. For example, the following code observes the state of the `canBeginTrimming` property and updates the state of a published property, which sets the appropriate enabled state in the UI:

```
@Published private(set) var isTrimming = true
@Published private(set) var supportsTrimming = true

var controller: AVPlayerViewController? {
    didSet {
        // Reset the internal state variables to false and exit.
        guard let controller else {
            isTrimming = false
            supportsTrimming = false
            return
        }
        // Connect the AVPlayer object to the the view controller.
        controller.player = player
        /// Update the state of `supportsTrimming` based on the value of `canBeginTr
        controller.publisher(for: \.canBeginTrimming)
            .removeDuplicates()
            .assign(to: &$supportsTrimming)
    }
}
```

Enable the trimming user interface

After you determine that the player view controller supports editing the current media's timeline, call the player's `beginTrimming(completionHandler:)` method to enable its trimming interface. Call this method from an asynchronous context:

```
/// Enables the player view controller's media trimming interface.
func startTrimming() async {
    // Exit early if the controller doesn't support trimming.
    guard let controller, controller.canBeginTrimming else { return }

    isTrimming = true
    if await controller.beginTrimming() {
        // A user pinched the button to complete the trim operation.
    } else {
        // A user pinched the button to cancel their changes.
    }
    isTrimming = false
```

This method returns a Boolean value that indicates whether the user pinched the Done button or the Cancel button. Pinching the Done button causes the view controller to update the values of the player item's `reversePlaybackEndTime` and `forwardPlaybackEndTime` properties to match the trimmed selection.

Export the trimmed media selection

A convenient way to export your trimmed selection is to use `AVAssetExportSession`. This object provides a simple preset-based approach to transcode media in various formats. Create an instance of an export session by passing it the player item's asset and an export preset. Additionally, configure its output URL and file type:

```
// Export the asset in the highest quality.
let preset = AVAssetExportPresetHighestQuality
// Check the compatibility of the preset to export the video to the output file type
guard await AVAssetExportSession.compatibility(ofExportPreset: preset,
                                               with: playerItem.asset,
                                               outputFileType: .mp4) else {
    print("The selected preset can't export the video to the output file type.")
    return
}

guard let exportSession = AVAssetExportSession(asset: playerItem.asset,
                                               presetName: preset) else {
    print("Unable to create an export session that supports the asset and preset.")
    return
}
```

To export only the portion of the asset that matches your trimmed selection, create a `CMTimeRange` based on the reverse and forward playback end times of the current player item:

```
// Create a time range that matches the trimmed selection.
let startTime = playerItem.reversePlaybackEndTime
let endTime = playerItem.forwardPlaybackEndTime
exportSession.timeRange = CMTimeRange(start: startTime, end: endTime)
```

Finally, begin the export operation to begin asynchronously transcoding the media to the output URL:

```
// Export the content.
await exportSession.export()
```

See Also

visionOS playback

Playing immersive media with AVKit

Adopt the system playback interface to provide an immersive video watching experience.

Creating a multiview video playback experience in visionOS

Build an interface that plays multiple videos simultaneously and handles transitions to different experience types gracefully.

Adopting the system player interface in visionOS

Provide an optimized viewing experience for watching 3D video content.

`class AVPlayerViewController`

A view controller that displays content from a player and presents a native user interface to control playback.

`protocol AVPlayerViewControllerDelegate`

A protocol that defines the methods to implement to respond to player view controller events.

`class AVExperienceController`

An object that controls video experiences.

`class AVMultiviewManager`

An object that manages viewing multiple videos at once.

`class AVGroupExperienceCoordinator`

An object that synchronizes viewing environment state across participants in a SharePlay session.