

[AVKit / Implementing Trimming in a macOS Player](#)

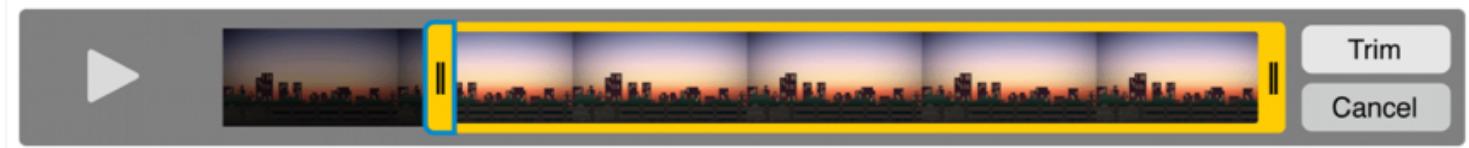
## Article

# Implementing Trimming in a macOS Player

Provide a QuickTime media-trimming experience in your macOS app.

## Overview

You use [AVPlayerView](#) to provide a playback experience like that of QuickTime Player in macOS. However, AVPlayerView not only provides the QuickTime playback interface, but it also provides the QuickTime media-trimming experience.



## Verify that Trimming Is Allowed

Before attempting to put the player into trimming mode, verify that trimming is allowed by querying the player view's [canBeginTrimming](#) property. This property returns `false` if you're playing an asset delivered over HTTP Live Streaming or if the asset is content protected. If you're presenting a menu item to initiate trimming, a good place to perform this check is in the [validateUserInterfaceItem\(\\_:\)](#) method of [NSDocument](#), so that the menu item can automatically be disabled if trimming is disallowed.

```
override func validateUserInterfaceItem(_ item: NSValidatedUserInterfaceItem) -> Bool {
    if item.action == #selector(beginTrimming) {
        return playerView.canBeginTrimming
    }
    return super.validateUserInterfaceItem(item)
}
```

# Enter Trimming Mode

After you've determined that the media supports trimming, you call the [beginTrimming\(completionHandler:\)](#). This method takes a completion block that you use to determine whether the user completed the trim or canceled the operation.

```
@IBAction func beginTrimming(_ sender: AnyObject) {
    playerView.beginTrimming { result in
        if result == .okButton {
            // user selected Trim button (AVPlayerViewTrimResult.okButton)...
        } else {
            // user selected Cancel button (AVPlayerViewTrimResult.cancelButton)...
        }
    }
}
```

## Transcode the Trimmed Asset

Because [AVAsset](#) is an immutable object, you may be wondering how its duration is changed when you click the Trim button. Trimming relies on a feature of [AVPlayerItem](#) to adjust the presented time range. [AVPlayerItem](#) provides the [reversePlaybackEndTime](#) and [forwardPlaybackEndTime](#) properties that set the in and out points for a media item. It doesn't change the underlying asset, but essentially changes your effective view of it. To save the results of the user's trim operation, you export a new copy of the asset, trimming it to the specified times. The simplest way to do this is to use [AVAssetExportSession](#), which provides a simple and performant way for you to transcode the media of an asset. You create a new export session, passing it the asset to export along with a transcoding preset to use.

```
// Transcoding preset
let preset = AVAssetExportPresetAppleM4V720pHD
let exportSession = AVAssetExportSession(asset: playerItem.asset, presetName: preset)
exportSession.outputFileType = AVFileTypeAppleM4V
exportSession.outputURL = // Output URL
```

This example uses a preset to export the media as a 720p, M4V file, but [AVAssetExportSession](#) supports a wide variety of export presets. To find out what export session presets are supported for the current asset, you can use the session's [exportPresets\(compatibleWith:\)](#) class method, passing it the asset you want to export. This method returns an array of valid presets that you can use in your export.

# Select the Trimmed Asset

To export only the content the user trimmed, you use the current player item's reverse and forward end-time values to define a [CMTimeRange](#) to set on the export session.

```
// Create CMTimeRange with the trim in/out point times
let startTime = self.playerItem.reversePlaybackEndTime
let endTime = self.playerItem.forwardPlaybackEndTime
let timeRange = CMTimeRangeFromTimeToTime(startTime, endTime)
exportSession.timeRange = timeRange
```

# Export the Trimmed Asset

To perform the actual export operation, you call its [exportAsynchronously\(completionHandler:\)](#) method. Check the status of the export session in the completion handler and handle completion and failure cases.

```
exportSession.exportAsynchronously {
    switch exportSession.status {
        case .completed:
            // Export Complete
        case .failed:
            // failed
        default:
            // handle others
    }
}
```

## See Also

### macOS playback and capture

`class AVPlayerView`

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

`class AVCaptureView`

A view that displays standard user interface controls for capturing media data.

