Article

Presenting Content Proposals in tvOS

Display a preview of an upcoming media item at the conclusion of the currently playing media item.

Overview

Media apps presenting serialized content, such as a TV show, often display a preview of the next episode in the series when you finish watching the current one. The user interface for this preview usually contains artwork and information about the proposed content. It also includes options for the user to either watch the next episode or return to the main menu. You add this functionality to your app using AVKit's content proposals.

Create a Content Proposal

You create a content proposal using the `AVContentProposal` class. This type models the data about the proposed content, such as its title, preview image, metadata, and content URL, and the time at which to present the proposal. You create and configure a content proposal instance as shown in the following code:

```
func makeProposal() -> AVContentProposal {

    // Present 10 seconds prior to the end of current presentation
    let time = currentAsset.duration - CMTime(value: 10, timescale: 1)
    let title = "My Show: Episode 2"
    let image = UIImage(named: "ms_ep2")!
    let proposal = AVContentProposal(contentTimeForTransition: time, title: title, p

    // Set custom metadata
    proposal.metadata = [
```

```
        makeMetadataItem(.commonIdentifierDescription, value: "Episode 2 Description
        makeMetadataItem(.iTunesMetadataContentRating, value: "TV-14"),
        makeMetadataItem(.quickTimeMetadataGenre, value: "Comedy")
    ]

    // Set content URL
    proposal.url = // The upcoming asset's URL

    // Return the created proposal.
    return proposal
}

private func makeMetadataItem(_ identifier: AVMetadataIdentifier, value: Any) -> AVM
    let item = AVMutableMetadataItem()
    item.identifier = identifier
    item.value = value as? NSCopying & NSObjectProtocol
    item.extendedLanguageTag = "und"
    return item.copy() as! AVMetadataItem
}
```

Create the Content Proposal's User Interface

In addition to defining your content proposal's data, you also need to create an interface to present this data to the user. You create this interface by subclassing the AVKit framework's `AVContentProposalViewController` class. At runtime, the system passes your subclass a reference to the current `AVContentProposal`, providing you the data to present. Your user interface should provide visual and descriptive information about the proposed content, and should also include options for the user to accept or reject the proposal.

When the system presents your proposal, it displays it over the currently playing full-screen video. You may want to scale this video to a smaller size so you can make more room to display the details of the proposed content. To do this, you override the view controller's `preferredPlayerViewFrame` property and return the desired video frame.

```
override var preferredPlayerViewFrame: CGRect {
    guard let frame = playerViewController?.view.frame else { return CGRect.zero }
    // Present the current video in a 960x540 window centered at the top of the wind
    return CGRect(x: frame.midX / 2.0, y: 0, width: 960, height: 540)
}
```

When the system presents the proposal, the player's view automatically animates to the specified frame.

> **Note**
>
> To lay out your content relative to the newly sized and positioned video frame, you use the `UILayoutGuide` provided by the view controller's `playerLayoutGuide` property.

Add Controls to the Content Proposal

Your presented user interface should also provide controls so the user can accept or reject the proposal. The event handlers for these actions should call the controller's `dismissContentProposal(for:animated:completion:)` method, indicating the user's choice.

```
// Handle acceptance
@IBAction func acceptContentProposal(_ sender: AnyObject) {
    dismissContentProposal(for: .accept, animated: true)
}

// Handle rejection
@IBAction func rejectContentProposal(_ sender: AnyObject){
    dismissContentProposal(for: .reject, animated: true)
}
```

Make the Content Proposal Eligible to be Presented

To make your content proposal eligible for the system to present, set it as the `nextContentProposal` property value of the current `AVPlayerItem`. The following example shows how to configure this property value in a playback app that manages a queue of `Video` objects, which is a custom value type that models the data of an individual video in the queue. The example code creates the required playback objects, creates a new `AVContentProposal` for the next video in the queue, and sets the video as the player item's `nextContentProposal`.

```
func prepareToPlay() {
    // Associate the AVPlayer with the AVPlayerViewController
    playerViewController?.player = player
    playerViewController?.delegate = self

    // Create a new AVPlayerItem with the current video's URL
    let playerItem = AVPlayerItem(url: currentVideo.url)
    player.replaceCurrentItem(with: playerItem)

    // Create an AVContentProposal for the next video (if it exists)
```

```
    playerItem.nextContentProposal = makeContentProposal(for: currentVideo.nextVideo
    player.play()
}

func makeContentProposal(for video: Video?) -> AVContentProposal? {
    guard let video = video else { return nil }
    guard let currentAsset = player.currentItem?.asset else { return nil }

    // Start the proposal within presentationInterval of the asset's duration
    let time = currentAsset.duration - video.presentationInterval
    let title = video.title
    let image = video.image

    // Create a new content proposal with the time, title, and image
    let contentProposal = AVContentProposal(contentTimeForTransition: time, title: t

    // Set the content URL for the proposal
    contentProposal.url = video.url
    // Automatically accept the proposal 1 second from playback end time
    contentProposal.automaticAcceptanceInterval = -1.0
    return contentProposal
}
```

Present the Content Proposal

With the content proposal set as the player item's `nextContentProposal`, the next step is to implement the methods of the `AVPlayerViewControllerDelegate` protocol. You use these methods to define how the system presents the content proposal, as well as to handle the acceptance or rejection of the proposed content.

To present your custom view controller in response to a request to present the next content proposal, implement the `playerViewController(_:shouldPresent:)` method. In this method, you set an instance of your custom `AVContentProposalViewController` as the player view controller's `contentProposalViewController` property.

```
func playerViewController(_ playerViewController: AVPlayerViewController, shouldPres
    // Set the presentation to use on the player view controller for this content pr
    playerViewController.contentProposalViewController = NextVideoProposalViewContro
    return true
}
```

If the presented `AVContentProposal` provides a valid content URL, the player view controller can automatically handle its acceptance or rejection. However, if you need more control over the handling of these actions, implement the `playerViewController(_:didAccept:)` and `playerViewController(_:didReject:)` methods. For example, the following code implements the `playerViewController(_:didAccept:)`method to play the proposed video and create a new content proposal for the next video in the queue.

```
func playerViewController(_ playerViewController: AVPlayerViewController, didAccept
    guard let player = playerViewController.player, let url = proposal.url else { re
    guard let video = currentVideo.nextVideo else { return }
    currentVideo = video

    // Create a new player item using the content proposal's URL
    let playerItem = AVPlayerItem(url: url)
    player.replaceCurrentItem(with: playerItem)
    player.play()

    // Prepare the new player item's next content proposal (if it exists)
    playerItem.nextContentProposal = makeContentProposal(for: currentVideo.nextVideo
}
```

See Also

tvOS playback and capture

- Customizing the tvOS Playback Experience

  Adopt the latest features of the redesigned tvOS player user interface to provide a more streamlined way to watch your content.

- Presenting Navigation Markers

  Present navigation markers in the Chapters panel to help users quickly navigate your content.

- Working with Interstitial Content

  Present additional content alongside your main media presentation using HTTP Live Streaming support.

- Working with Overlays and Parental Controls in tvOS

  Add interactive overlays, parental controls, and livestream channel flipping using a player view controller.

{} Supporting Continuity Camera in your tvOS app

Capture high-quality photos, video, and audio in your Apple TV app by connecting an iPhone or iPad as a continuity device.

`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` `AVInterstitialTimeRange`

A time range in an audiovisual presentation for content with an interstitial designation, such as advertisements or legal notices.

`class` `AVNavigationMarkersGroup`

A set of markers for navigating playback of an audiovisual presentation.

`class` `AVContentProposalViewController`

A view controller that proposes content to watch next.

`class` `AVDisplayManager`

A tvOS management object that controls whether a TV switches modes to match the video's native mode.

`class` `AVContinuityDevicePickerViewController`

A view controller that provides an interface to a person so they can select and connect a continuity device to the system.

`protocol` `AVContinuityDevicePickerViewControllerDelegate`

An interface that responds to events from a continuity device picker view controller.