Article

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.

Overview

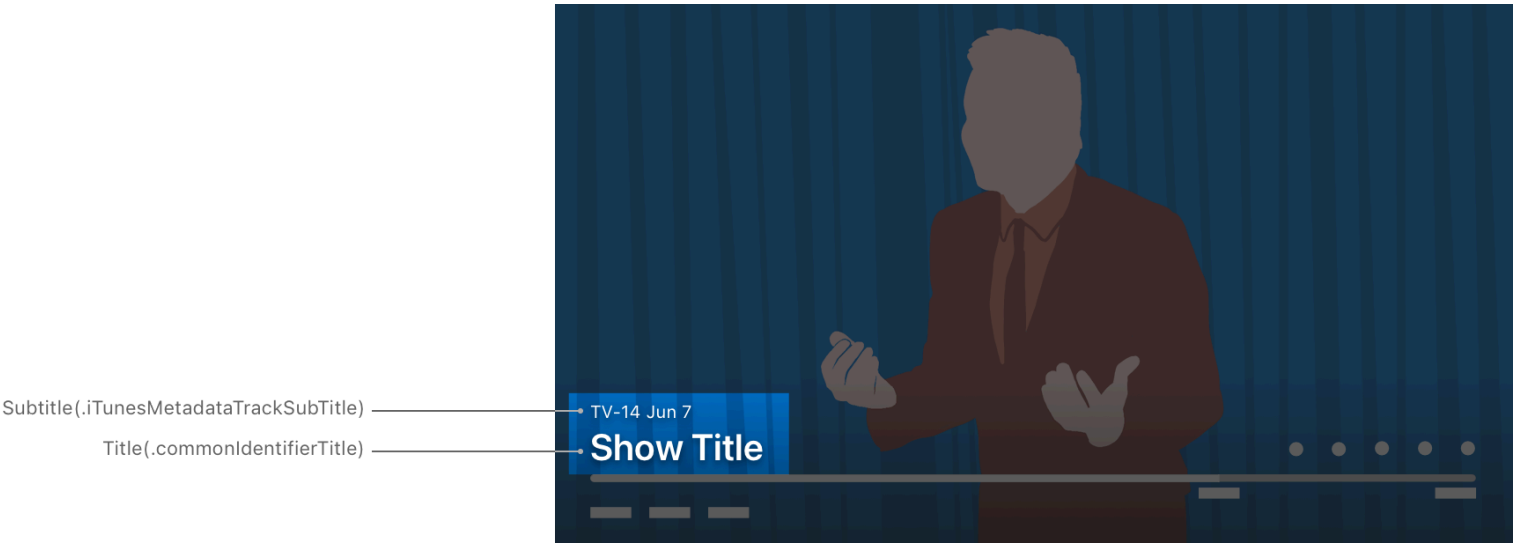
tvOS 15 introduces an all-new playback experience that helps viewers stay in the moment by providing a more consistent, useful, and interactive user interface. The redesigned UI provides viewers with convenient access to relevant controls and information while always keeping the focus on the content.

It's simple to provide this playback experience in your app by using `AVPlayerViewController`. Using this class for your player UI provides your app the same features and familiar interactions found in the Apple TV app, including support for voice commands using the Siri Remote and presenting video in Picture in Picture. In tvOS 15, `AVPlayerViewController` is more configurable and customizable than ever, giving you new ways to tailor your app's video playback experience.

> **Note**
>
> Existing apps that use `AVPlayerViewController` adopt the new styling and features of the redesigned player UI when they link against the tvOS 15 SDK.

Display Supporting Metadata

The player user interface displays a title view above the transport bar when the current player item contains title and subtitle metadata. When playing live streaming content, the title view may also display a badge to indicate that content state to the viewer.



The title view retrieves the values it displays from an asset's `commonIdentifierTitle` and `iTunesMetadataTrackSubTitle` metadata items, when available. If the media your app plays doesn't contain embedded metadata, you can add custom metadata by creating instances of `AVMetadataItem`. The table below lists the metadata values that the player user interface supports.

Metadata	Identifier	Type
Title	`commonIdentifierTitle`	`Data`
Subtitle	`iTunesMetadataTrackSubTitle`	`String`

Metadata	Identifier	Type
Artwork	commonIdentifierArtwork	String
Description	commonIdentifierDescription	String
Genre	quickTimeMetadataGenre	String
Content rating	iTunesMetadataContentRating	String

In an app that defines a structure that stores simple string-based metadata values, you could map its values to their appropriate metadata identifiers and build an array of metadata items as shown below.

```
func createMetadataItems(for metadata: Metadata) -> [AVMetadataItem] {
    let mapping: [AVMetadataIdentifier: Any] = [
        .commonIdentifierTitle: metadata.title,
        .iTunesMetadataTrackSubTitle: metadata.subtitle,
        .commonIdentifierArtwork: UIImage(named: metadata.image)?.pngData() as Any,
        .commonIdentifierDescription: metadata.description,
        .iTunesMetadataContentRating: metadata.rating,
        .quickTimeMetadataGenre: metadata.genre
    ]
    return mapping.compactMap { createMetadataItem(for:$0, value:$1) }
}

private func createMetadataItem(for identifier: AVMetadataIdentifier,
                                value: Any) -> AVMetadataItem {
    let item = AVMutableMetadataItem()
    item.identifier = identifier
    item.value = value as? NSCopying & NSObjectProtocol
    // Specify "und" to indicate an undefined language.
    item.extendedLanguageTag = "und"
    return item.copy() as! AVMetadataItem
}
```

To apply the metadata to the current player item, set the array of metadata items as the value of the player item's externalMetadata property.

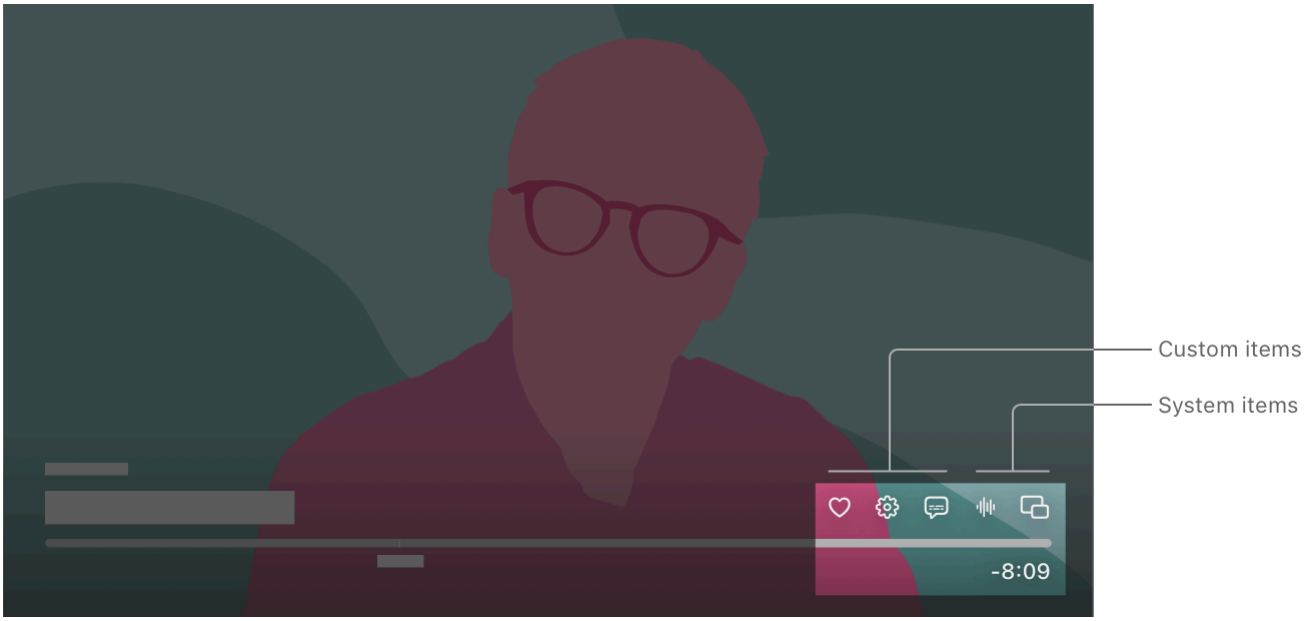
```
let metadata: Metadata = // A structure that contains simple string values.
playerItem.externalMetadata = createMetadataItems(for: metadata)
```

Only the title and subtitle values display in the title view. The player presents the other supported metadata values in its Info tab, which the Display Content Tabs section below describes.

> **Note**
>
> If your app would prefer to always hide the title view, set the value of the player view controller's `transportBarIncludesTitleView` to `false`.

Add Custom Transport Bar Items

The redesigned transport bar displays controls along its trailing side, which provides viewers quick access to common actions. The system automatically displays controls to configure common playback settings, like selecting subtitles and enabling Picture in Picture, but apps can also add custom controls to the transport bar.



You define custom transport bar items by creating instances of `UIAction` or `UIMenu`. For example, to create a simple control that toggles whether the current movie is a favorite, you could create an instance of `UIAction` as shown below.

```
// Create ♡ and ♥ images.
let heartImage = UIImage(systemName: "heart")
let heartFillImage = UIImage(systemName: "heart.fill")

// Create an action to add the item to the viewer's favorites.
let stateImage = isFavorited ? heartFillImage : heartImage
let favoriteAction = UIAction(title: "Favorites",
                              image: stateImage) { [weak self] action in
```

```
    // Add the movie to or remove it from the viewer's favorites list.
    self?.isFavorited.toggle()

    // Update the button image to reflect the new state.
    action.image = isFavorite ? heartFillImage : heartImage
}
```

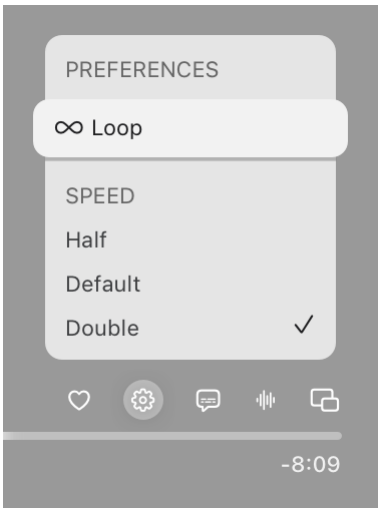
You can also create more advanced arrangements of actions and present them in a menu. The following example creates a custom preferences menu that adds a gear icon to the transport bar. When a user clicks the icon, the system presents a pop-up menu with an option to enable looping playback and an inline submenu to control playback speed.

```
// Create ∞ and ⚙ images.
let loopImage = UIImage(systemName: "infinity")
let gearImage = UIImage(systemName: "gearshape")

// Create an action to enable looping playback.
let loopAction = UIAction(title: "Loop", image: loopImage, state: .off) { action in
    action.state = (action.state == .off) ? .on : .off
}

let speedActions = ["Half": 0.5, "Default": 1.0, "Double": 2.0].map { title, value i
    UIAction(title: title, state: self.playbackSpeed == value ? .on : .off) { [weak
        // Update the current playback speed.
        self?.playbackSpeed = value
        action.state = .on
    }
}

// Create the submenu.
let submenu = UIMenu(title: "Speed",
                     options: [.displayInline, .singleSelection],
                     children: speedActions)

// Create the main menu.
let menu = UIMenu(title: "Preferences", image: gearImage, children: [loopAction, sub
```

To have the player view controller present the custom items, set them as the player view controller's `transportBarCustomMenuItems` property value.

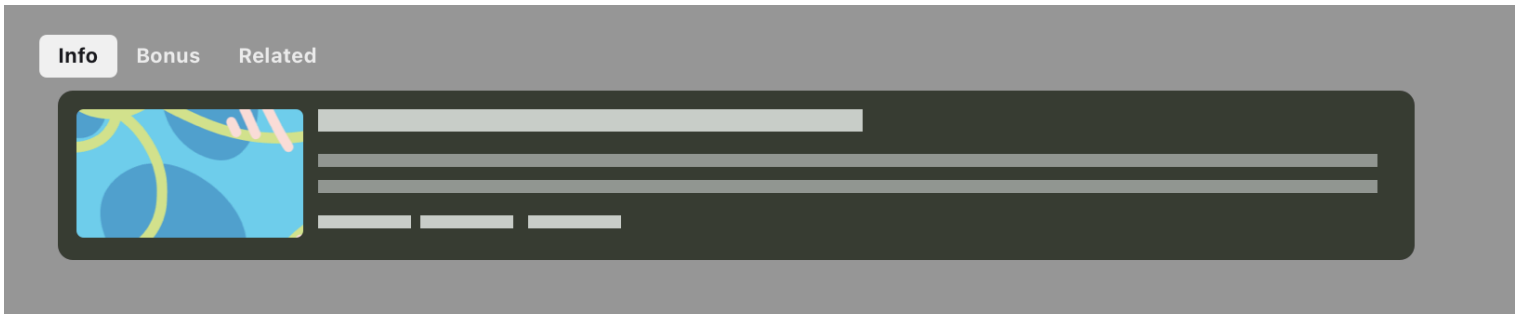
```
// Set the custom transport bar items.
playerViewController.transportBarCustomMenuItems = [favoriteAction, menu]
```

Setting the custom action and menu adds two new items to the transport bar.



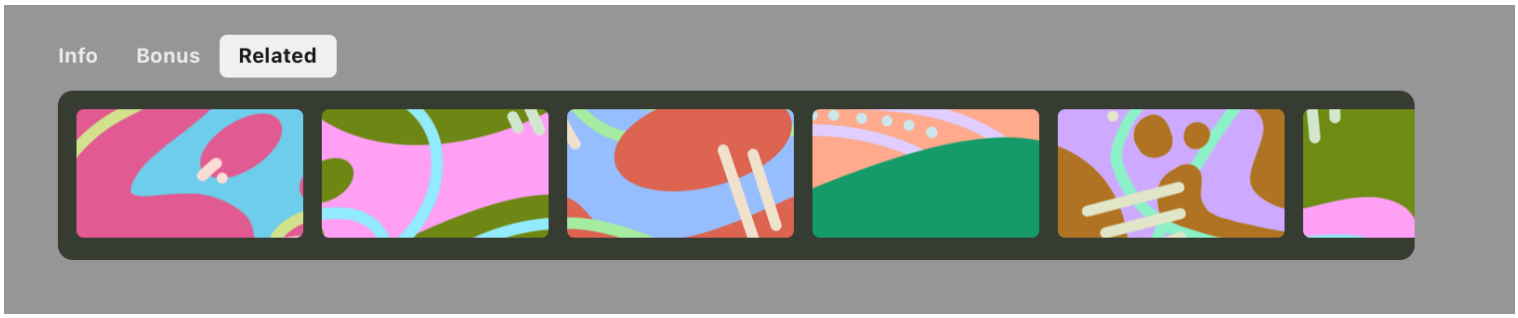
Display Content Tabs

The tvOS player UI can display one or more content tabs below the transport bar to show supporting information or related content. By default, the player presents an Info tab when an asset contains embedded metadata or when you set external metadata on the player item, as the Display Supporting Metadata section above describes.



Your app can also define custom tabs by creating subclasses of `UIViewController` to present your supporting content, and set them as the value of the player view controller's `customInfoViewControllers` property. The system uses the custom view controller's `title` property value as the tab title, so initialize it before setting it on the player view controller.

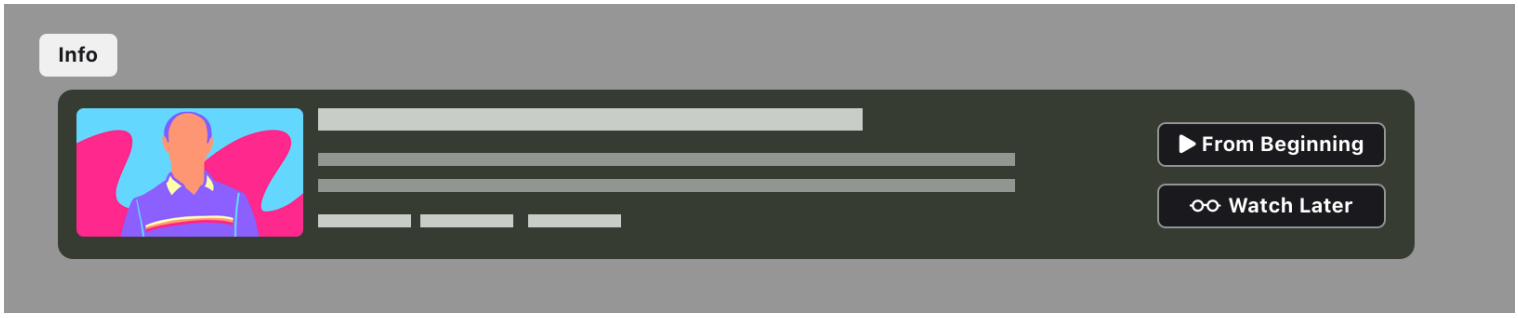
```
// Set custom content tabs on the player UI.
playerViewController.customInfoViewControllers = [
    BonusContentViewController(title: "Bonus"),
    RelatedContentViewController(title: "Related")
]
```



For the player view controller to size your content appropriately, specify a `preferredContentSize` or define appropriate auto layout constraints. The system sizes all view controllers to the height of the tallest content tab, so size your custom view controllers consistently or verify that they lay out as you expect at their runtime height.

Present Actions in the Info Tab

A player view controller presents an Info tab when playing an asset with embedded or external metadata. The tab's view displays the metadata details, and it may show up to two `UIAction` controls along its trailing edge, as shown below.



You customize the actions the view presents by setting a value for the player view controller's `infoViewActions` property. When playing nonlive content, this property contains a single-element array that presents an action to play the content from the beginning. You may replace the default value (if present), add an additional action, or set this property value to an empty array to display no actions. The example below shows how to add a Watch Later action to the view.

```
let glasses = UIImage(systemName: "eyeglasses")
let watchLater = UIAction(title: "Watch Later", image: glasses) { action in
    // Add or remove the item from the user's watch list,
    // and update the action state accordingly.
}
// Append the action to the array.
playerViewController.infoViewActions.append(watchLater)
```

Present Actions Contextually

You can use the tvOS player UI to present controls contextually, which you display for a specific range of time in the content and then dismiss. A common use for this type of control is a Skip button that displays during the title sequence of a movie or TV show. Clicking the button allows viewers to bypass the introduction and quickly skip to the main content.

`AVPlayerViewController` provides a `contextualActions` property that you can use to specify one or more actions to present. The player displays them along the bottom-trailing side of the screen. The following code example shows a simple implementation of an action that seeks the player forward to the time of the main content.

```
// Define an action to skip the introduction of a media asset.
private lazy var skipAction = UIAction(title: "Skip") { [weak self] _ in
    guard let self = self else { return }
    self.player.seek(to: self.skipToTime)
}
```

When you set a value for the `contextualActions` property, the player presents the controls immediately. To present them only during a relevant section of the content, observe the player timing by adding a periodic or boundary time observer. The following example defines a periodic time observer that fires every second during normal playback. In each invocation, it evaluates the new time to determine if it falls within the presentation range. If it does, the example sets the skip action as the contextual actions value; otherwise, it clears the value by setting it to an empty array.

```
func addTimeObserver() {
    // Observe the player's timing every second.
    let interval = CMTime(value: 1, timescale: 1)
```

```
    timeObserver = player.addPeriodicTimeObserver(forInterval: interval,
                                                  queue: .main) { [weak self] time i
        guard let self = self else { return }
        // If the time is within the defined skip range, present the skip actions.
        let actions = self.skipRange.containsTime(time) ? [self.skipAction] : []
        self.playerViewController.contextualActions = actions
    }
}
```

See Also

tvOS playback and capture

- 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.

- Presenting Content Proposals in tvOS

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

- {} 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.