Class

ASAccessorySession

A class to coordinate accessory discovery.

iOS 18.0+ | iPadOS 18.0+

```
class ASAccessorySession
```

Mentioned in

Discovering and configuring accessories

Overview

Use an instance of `ASAccessorySession` to interact with the AccessorySetupKit framework.

Start the session by calling `activate(on:eventHandler:)`, and pass in a dispatch queue and an event-handling closure. AccessorySetupKit calls back to your event handler as the discovery session processes events.

With your event-handler prepared, create an array of `ASPickerDisplayItem` instances to describe accessories your app can set up. Pass this array to the session's `showPicker(for:completionHandler:)` method to allow someone using your app to choose a discovered accessory to set up. Your event handler receives events as the picker appears and dismisses, as well as when the person using the app adds an accessory.

Important

Starting in iOS 18.4, apps can use AccessorySetupKit for discovery and setup of Bluetooth LE devices that conform to the Human Interface Device (HID) service, such as keyboard and mouse accessories. The HID accessory needs to advertise a custom service besides the HID service. Add the `bluetoothHID` option to the `supportedOptions` and configure the `ASDiscoveryDescriptor` to discover the custom service instead of the HID service.

Topics

Managing the session life cycle

```
func activate(on: dispatch_queue_t, eventHandler: (ASAccessoryEvent) ->
Void)
```

Activate the session and start delivering events to an event handler.

```
func invalidate()
```

Invalidate the session by stopping any operations.

Displaying an accessory picker

```
func showPicker(completionHandler: ((any Error)?) -> Void)
```

Present a picker that shows accessories managed by a Device Discovery Extension in your app.

```
func showPicker(for: [ASPickerDisplayItem], completionHandler: ((any
Error)?) -> Void)
```

Present a picker that shows discovered accessories matching an array of display items.

Customizing picker behavior

```
var pickerDisplaySettings: ASPickerDisplaySettings?
```

Settings that affect the display of the accessory picker.

```
class ASPickerDisplaySettings
```

A type that contains settings to customize the display of the accessory picker

Updating the picker

```
func updatePicker(showing: [ASDiscoveredDisplayItem], completionHandler
: ((any Error)?) -> Void)
```

Updates the picker with app-filtered accessories.

Ending filtered discovery

```
func finishPickerDiscovery(completionHandler: ((any Error)?) -> Void)
```

Finish the discovery session in the picker and show a timeout error.

Accessing discovered accessories

```
var accessories: [ASAccessory]
```

An array of previously-selected accessories for this application.

Managing accessories

```
func renameAccessory(ASAccessory, options: ASAccessory.RenameOptions,
completionHandler: ((any Error)?) -> Void)
```

Displays a view to rename an accessory.

```
struct RenameOptions
```

Options that affect the behavior of an accessory renaming operation.

```
func removeAccessory(ASAccessory, completionHandler: ((any Error)?) ->
Void)
```

Removes an accessory.

Managing authorization

```
func finishAuthorization(for: ASAccessory, settings: ASAccessory
Settings, completionHandler: ((any Error)?) -> Void)
```

Finish authorization of a partially-setup accessory.

```
class ASAccessorySettings
```

Properties of an accessory.

```
func failAuthorization(for: ASAccessory, completionHandler: ((any Error
)?) -> Void)
```

End authorization of a partially-configured accessory as a failure.

```
func updateAuthorization(for: ASAccessory, descriptor: ASDiscoveryDescriptor, completionHandler: ((any Error)?) -> Void)
```

Displays a view to upgrade an accessory with additional technology permissions.

Relationships

Inherits From

`NSObject`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSObjectProtocol`
`Sendable`
`SendableMetatype`

See Also

Essentials

{} Setting up and authorizing a Bluetooth accessory

Discover, select, and set up a specific Bluetooth accessory without requesting permission to use Bluetooth.

Discovering and configuring accessories

Detect nearby accessories and facilitate their setup.