☰   ∞ **Meta Wearables**   🔍   Login   Register

API reference   ⧉

WearablesInterface Protocol

The primary interface for Wearables Device Access Toolkit.

Signature 🔗

```
extension WearablesInterface
```
⧉

Properties 🔗

devices [Get]	The current list of devices available.
registrationState [Get]	The current registration state of the user's devices. See [RegistrationState](#) for options.

Functions 🔗

addDeviceSessionStateListener (forDeviceId , listener)	Adds a listener to receive callbacks when the session state changes for a specific device. The listener is immediately called with the current session state.

Signature

```
public func
addDeviceSessionStateListener(
forDeviceId: DeviceIdentifier,
listener: @escaping (SessionState) ->
Void) -> AnyListenerToken
```
⧉

Parameters

callback to execute when the session state changes.

Returns

`AnyListenerToken` A token that can be used to cancel t' ⌐
listener. When the token deinits the listener is also cance'

**addDevicesListe
ner** (listener)

Adds a listener to receive callbacks when the device list
changes. The listener is immediately called with the current
devices.

Signature

```
public func addDevicesListener(_
listener: @escaping ([DeviceIdentifier])
-> Void) -> AnyListenerToken
```

Parameters

`_ listener: @escaping ([DeviceIdentifier]) ->
Void` The callback to execute when the device list changes.

Returns

`AnyListenerToken` A token that can be used to cancel the
listener. When the token deinits the listener is also canceled.

**addRegistration
StateListener** (
listener)

Adds a listener to receive callbacks when the registration
state changes. The listener is immediately called with the
current state.

Signature

```
public func
addRegistrationStateListener(_ listener:
@escaping (RegistrationState) -> Void) -
> AnyListenerToken
```

Parameters

`_ listener: @escaping (RegistrationState) ->
Void` The callback to execute when the registration state
changes.

Returns

☰   ∞ Meta Wearables   🔍   Login   Register

---

checkPermission
Status (
permission)

Checks if a specific permission is granted for the current application.

Signature

```
public func checkPermissionStatus(_
permission: Permission) ->
PermissionStatus
```

Parameters

_ permission: Permission The type of permission to check.

Returns

PermissionStatusPermissionStatus The status of the permission.

Throws

PermissionError if the operation fails.

---

deviceForIdentif
ier (identifier)

Fetch the underlying Device object for a given DeviceIdentifier.

Signature

```
public func deviceForIdentifier(_
identifier: DeviceIdentifier) -> Device?
```

Parameters

_ identifier: DeviceIdentifier The device identifier to fetch.

Returns

Device? The Device object for the given device identifier.

---

devicesStream ()

Creates an AsyncStream for observing device list changes.

Signature

Returns

```
AsyncStream<[DeviceIdentifier]>
```

handleUrl (url)

Handles callback URLs from the Meta AI app during registration and permission flows.

This method must be called when your app receives a URL callback after the user completes an action in the Meta AI app. This includes callbacks from startRegistration(), startUnregistration(), and permission requests.

The SDK will determine if the URL is relevant to the Wearables Device Access Toolkit. If not relevant, the method returns `false` without throwing an error.

Platform Flow On iOS, the Meta AI app returns to your app via a URL scheme callback. You must: 1. Configure your app's URL schemes in Info.plist 2. Implement URL handling in your app delegate or scene delegate 3. Call this method with the received URL

Signature

```swift
public func handleUrl(_ url: URL) ->
Bool
```

Parameters

`_ url: URL` The incoming URL to handle.

Returns

`Bool` `true` if the URL was handled by the Wearables Device Access Toolkit, `false` if it's not relevant to the Wearables Device Access Toolkit.

Throws

RegistrationError if there is an error processing a relevant URL.

registrationStat eStream ()

Creates an `AsyncStream` for observing registration state changes.

Signature

☰   ∞ **Meta Wearables**   🔍   Login   Register

Returns

`AsyncStream<`[`RegistrationState`](#)`>`

requestPermissi on (permission)	Requests a specific permission on AI glasses.
	This method opens the Meta AI app where the user completes the permission request flow. After the user responds in the Meta AI app, your app will receive a callback URL that must be passed to [handleUrl(_:)](#) to complete the permission request.
	
	Signature
	
	<pre>public func requestPermission(_
	permission: Permission) ->
	PermissionStatus</pre> ⎘
	
	Parameters
	
	`_ permission:` [`Permission`](#) The type of permission to request.
	
	Returns
	
	[`PermissionStatus`](#) The [`PermissionStatus`](#) after the user responds.
	
	Throws
	
	[`PermissionError`](#) if there is an error starting the permission request process.

startRegistratio n ()	Initiates the registration process with AI glasses. This method opens the Meta AI app where the user completes the registration flow. After the user completes the flow in the Meta AI app, your app will receive a callback URL that must be passed to [handleUrl(_:)](#) to complete the registration. The [registrationState](#) property will be updated throughout the registration process. **Signature** <pre>public func startRegistration()</pre> ⎘

☰   ∞ Meta Wearables   🔍   Login   Register

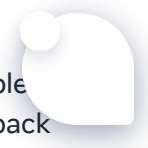|	

startUnregistrati on ()	Initiates the unregistration process with AI glasses. This method opens the Meta AI app where the user completes the unregistration flow. After the user comple the flow in the Meta AI app, your app will receive a callback URL that must be passed to [handleUrl(_:)](#) to complete the unregistration.
	The [registrationState](#) property will be updated throughout the unregistration process.
	
	Signature
	
	``swift
	public func startUnregistration()
	``
	
	Throws
	
	[RegistrationError](#) if there is an error starting the unregistration process.

∞ Meta Wearables

Build with Meta

Social Technologies

Meta Horizon

AI

Worlds

Wearables

About us

Careers

Research

Products

Support and legal

Wearables Developer Terms

Acceptable Use Policy

Legal

Privacy

Meta Wearables