Article

Supporting Single Sign-On in a Web Browser App

Extend your web browser app to handle web authentication requests from other apps.

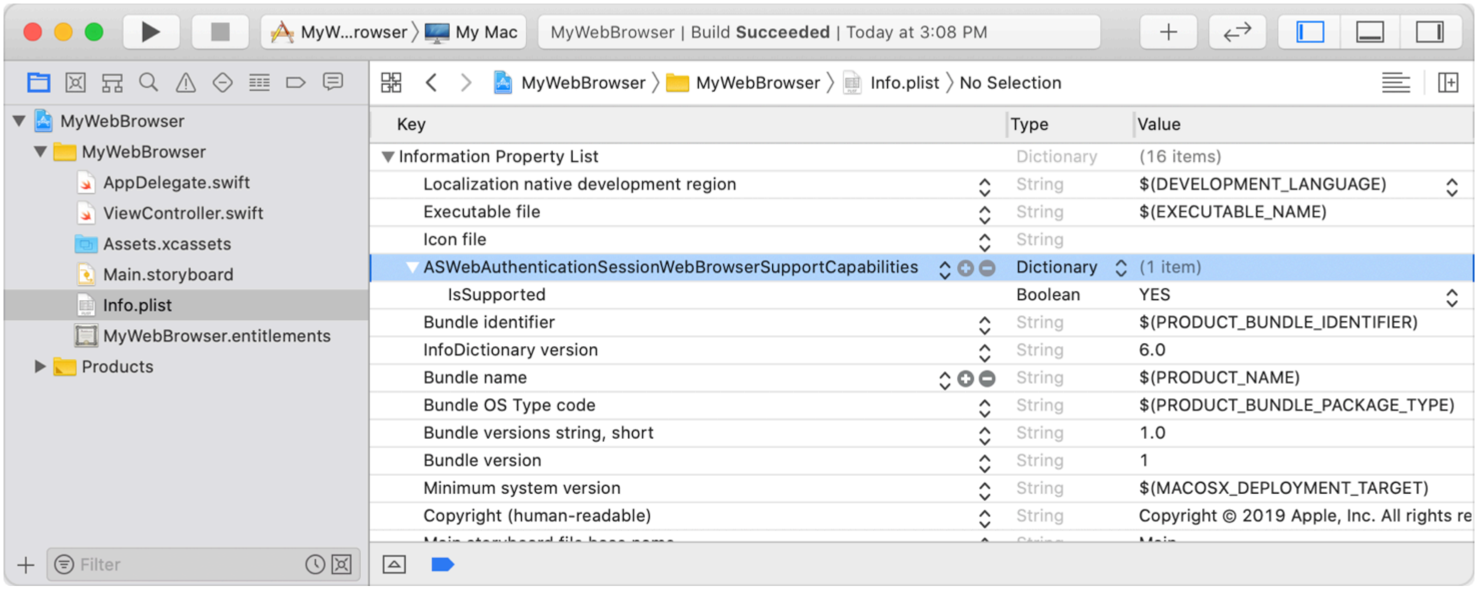
Overview

Apps can authenticate users through a web service using an instance of `ASWebAuthenticationSession`. When an app calls the authentication session's `start()` method, the system asks the user's default web browser to initiate the authentication attempt at a given URL. If the default browser doesn't handle authentication requests, the system falls back on Safari. Either way, the designated browser loads the URL, waits for the user to authenticate, and returns a callback URL that indicates the outcome of the attempt.

If you distribute a web browser app on macOS, you can make it eligible to participate in this flow. You register the browser as a session handler, and then listen for and handle authentication requests.

Declare the Session Handling Capability

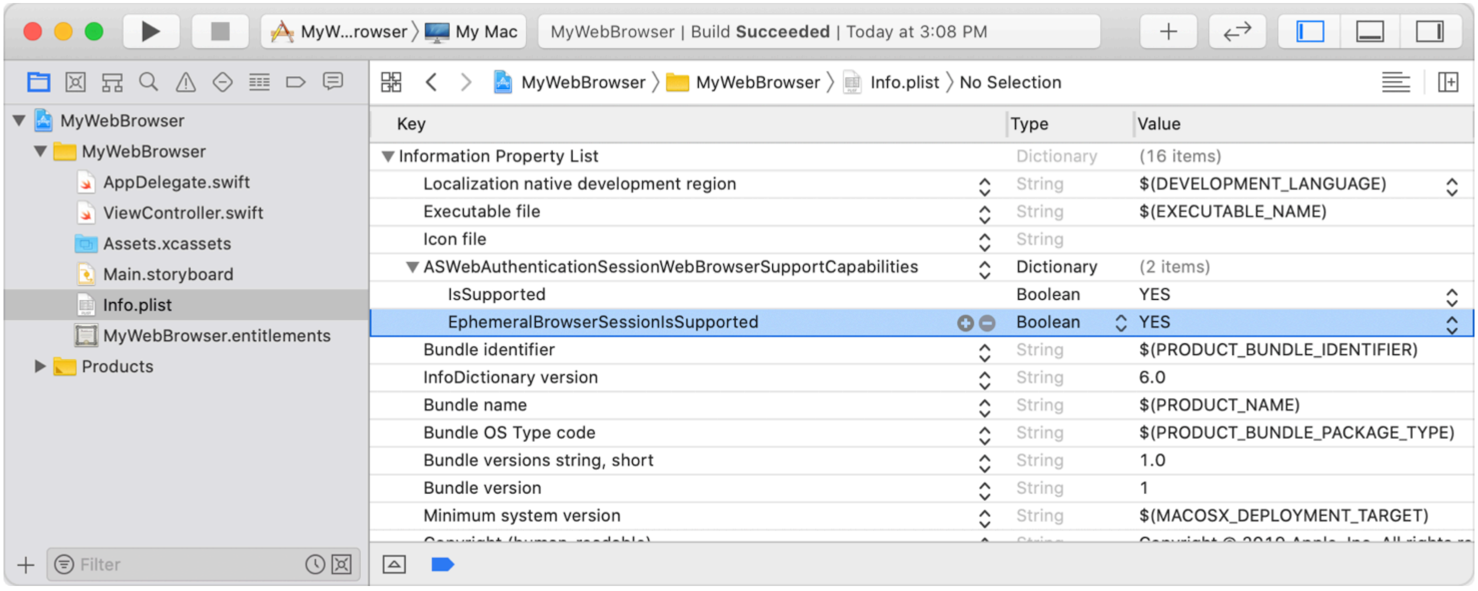
Using the Xcode property list editor, add the `ASWebAuthenticationSessionWebBrowserSupportCapabilities` key to your web browser's `Information Property List`. For the key's value, create a dictionary that contains the `IsSupported` key, with a corresponding value of `YES`.



Key	Type	Value
Information Property List	Dictionary	(16 items)
Localization native development region	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Icon file	String	
ASWebAuthenticationSessionWebBrowserSupportCapabilities	Dictionary	(1 item)
IsSupported	Boolean	YES
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Minimum system version	String	$(MACOSX_DEPLOYMENT_TARGET)
Copyright (human-readable)	String	Copyright © 2019 Apple, Inc. All rights re

By declaring this capability, you tell the system that your web browser app handles single sign-on requests. If the user has set your browser as the default, the system routes authentication requests to it.

Optionally, add the `EphemeralBrowserSessionIsSupported` key with a value that indicates whether your browser supports ephemeral browsing.



Key	Type	Value
Information Property List	Dictionary	(16 items)
Localization native development region	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Icon file	String	
ASWebAuthenticationSessionWebBrowserSupportCapabilities	Dictionary	(2 items)
IsSupported	Boolean	YES
EphemeralBrowserSessionIsSupported	Boolean	YES
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Minimum system version	String	$(MACOSX_DEPLOYMENT_TARGET)

If you don't provide the key, or if you set its value to NO and an app tries to conduct an ephemeral authentication session, the system warns the user. If do you declare support by setting the value to YES, be sure to respect the `shouldUseEphemeralSession` property on any incoming authentication requests, as described below in Perform Authentication.

> **Note**
>
> It's strongly recommended that your web browser support ephemeral sessions. Apps can specifically request this kind of session, and it's important to honor the request.

Listen for Authentication Requests

Adopt the ASWebAuthenticationSessionWebBrowserSessionHandling protocol in your web browser app to receive authentication requests. Choose a class that can act as the handler, and declare conformance to the protocol:

```
class MyAuthenticationHandler: ASWebAuthenticationSessionWebBrowserSessionHandling {
    var request: ASWebAuthenticationSessionRequest?
}
```

From within the conforming class, implement the protocol's begin(_:) method to receive new ASWebAuthenticationSessionRequest instances. Handle the request using the data it encapsulates as described in Perform Authentication below.

```
func begin(_ request: ASWebAuthenticationSessionRequest!) {
    self.request = request   // Store for later.

    if request.shouldUseEphemeralSession == true {
        // Load request.url in an isolated session and wait for the callback.
    } else {
        // Load request.url and wait for the callback.
    }
}
```

Implement the cancel(_:) method to listen for cancellations, which might happen if the calling app terminates or cancels the operation. If you have multiple requests in progress, you can use the cancellation request's uuid property to identify the in-progress request to cancel:

```
func cancel(_ request: ASWebAuthenticationSessionRequest!) {
    // Abandon the request and clean up.
    self.request = nil
}
```

After implementing the interface, tell the system how to find your session handler by setting the sessionHandler property of the shared session manager. You typically do this once at startup. For example, you might set the property in your app delegate's applicationDidFinishLaunching(_:) method:

```
func applicationDidFinishLaunching(_ aNotification: Notification) {
    let manager = ASWebAuthenticationSessionWebBrowserSessionManager.shared
    manager.sessionHandler = MyAuthenticationHandler()
    // The manager keeps a strong reference to your handler.
}
```

Perform Authentication

When your handler receives a new authentication request, load the URL given in the request's `url` property and display the content to the user. The user interacts with the content to authenticate—for example, by entering credentials and clicking a button. The service performing the authentication indicates the outcome by redirecting the browser to a URL that uses a known callback scheme.

The request's `callbackURLScheme` property tells your browser what callback scheme the service uses. When your browser detects a redirect involving this scheme, pass the entire URL back to the session manager by calling the request's `complete(withCallbackURL:)` method. For example, when using the WebKit API, you can do this from the navigation delegate's `webView(_:decidePolicyFor:decisionHandler:)` method:

```
func webView(_ webView: WKWebView,
             decidePolicyFor navigationAction: WKNavigationAction,
             decisionHandler: @escaping (WKNavigationActionPolicy) -> Void) {

    guard let url = navigationAction.request.url else { return }

    if url.scheme == request?.callbackURLScheme {
        request?.complete(withCallbackURL: url)
    } else {
        // Handle normally.
    }
}
```

Alternatively, if the browser can't complete the operation for any reason—for example, because the user has closed the authentication window—call the request's `cancelWithError(_:)` method instead.

If you declare that your browser supports ephemeral browsing, as described above in Declare the Session Handling Capability, be sure to respect the request's `shouldUseEphemeralSession` property. When this value is `true`, avoid using any existing browsing data, like cookies, during the authentication process. Also, avoid retaining any data collected during the authentication attempt beyond the lifetime of the attempt, or sharing it with any other session.

Handle Launch for Authentication

When your browser supports the authentication flow, the system might launch the browser specifically for the purpose of authentication. You can detect this condition by checking the `wasLaunchedByAuthenticationServices` property of the shared session manager.

```
func applicationDidFinishLaunching(_ aNotification: Notification) {
    let manager = ASWebAuthenticationSessionWebBrowserSessionManager.shared
    manager.sessionHandler = MyAuthenticationHandler()

    if manager.wasLaunchedByAuthenticationServices == true {
        // Adjust startup behavior accordingly.
    }
}
```

Use this indicator to adjust the behavior of your browser at startup. For example, you might avoid restoring previously open windows or tabs.

See Also

Web authentication sessions

Authenticating a User Through a Web Service

Use a web authentication session to authenticate a user in your app.

Securing Logins with iCloud Keychain Verification Codes

Use time-based codes generated on-device for a secure authentication experience.

`class ASWebAuthenticationSession`

A session that an app uses to authenticate a user through a web service.

`struct WebAuthenticationSession`

A SwiftUI environment value that views use to authenticate someone using a web service.

`class ASWebAuthenticationSessionWebBrowserSessionManager`

A session manager that mediates sharing data between an app and a web browser.

`ASWebAuthenticationSessionWebBrowserSupportCapabilities`

A collection of keys that a browser app uses to declare its ability to handle authentication requests from other apps.