Article

Signing people in to their media accounts automatically

Implement single sign-on for media-streaming apps by managing a sign-in token on a person's Apple Account.

Overview

The Automatic Sign-In feature offers a single sign-on experience for media-streaming apps that have the `com.apple.smoot.subscriptionservice` entitlement. The first time a person signs in to your media streaming service, your app prompts them to opt in to Automatic Sign-In. If they agree, your app generates a *string value* token. You provide the token to the framework, which stores it on the person's Apple Account to sign the person in across all of their devices. When your app launches again, the framework provides the token if it exists on the person's Apple Account, which your app uses to bypass the sign-in prompt and give the person quick access to the media streaming features in your app.

In addition, the feature creates a single sign-on experience when a person:

- Acquires a new Apple TV, iPhone, or iPad and restores app purchases
- Downloads your media-streaming app later on another device

Only your app interprets the value to authenticate the person, for example, by ensuring it represents an account with a valid subscription. The framework only manages *token storage* for your app. In addition to storing the token value on the person's Apple Account, the framework also records the token *authorization*, that is, the choice a person makes on the prompt. If the person dismisses the prompt without approving, your app proceeds to enable streaming without creating a token value.

Important

If your app supports tvOS and other platforms, including iOS and iPadOS, use Automatic Sign-In to share a unified code base on Apple platforms for your media streaming apps. Because iCloud Keychain isn't available on tvOS, Automatic Sign-In provides a cross-platform solution to implement single sign-on for iOS, iPadOS, and tvOS apps.

Automatic Sign-In doesn't replace iCloud Keychain for storing generic secrets, such as those that enable a person to manage their account, or view their personal information.

Generate an Automatic Sign-In token

When your app launches with the person signed out, check for an Automatic Sign-In token (`autoSignInToken`) to use for signing them in.

If the token is undefined (`value` is `nil` or `authorization` is `VSUserAccountManager.AutoSignInAuthorization.notDetermined`), prompt the person to sign in to your media streaming service:

```
// Access the static shared user account manager.
let accountManager = VSUserAccountManager.shared
let autoSignInToken = try? await accountManager.autoSignInToken

if autoSignInToken?.value == nil || autoSignInToken?.authorization == .notDetermined
    // Prompt the person to sign in.
```

After the person signs in successfully, prompt them to opt in to Automatic Sign-In. Call `requestAutoSignInAuthorization()` to present a sheet that asks for their approval:

```
// Media streaming service sign-in success:
if case .success = newSignInResult {
    // Prompt to opt-in to Automatic Sign-In.
    if let context = try? await accountManager.requestAutoSignInAuthorization() {
        // Generate the token.
    }
}
```

If the person dismisses the prompt without approving, the system leaves the `authorization` status `VSUserAccountManager.AutoSignInAuthorization.notDetermined` and the

framework doesn't prompt them again until the app calls `requestAutoSignInAuthorization()` once more.

If the person approves the prompt, the method returns an `VSUserAccountManager.AutoSignInTokenUpdateContext` instance with an `authorization` set to `VSUserAccountManager.AutoSignInAuthorization.granted`. Generate a token `value` and pass it to the framework using `updateAutoSignInToken(_:updateContext:)`:

```
let newToken = await generateNewAutoSignInToken() // Your implementation.
try? await accountManager.updateAutoSignInToken(newToken, updateContext: context)
```

The system stores the token value and its authorization status to the person's Apple Account, where the framework can find it from your app across the their other devices.

Sign in with an Automatic Sign-In token

If your app launches signed out but the Automatic Sign-In token (`autoSignInToken`) has a valid `value`, consume the token and sign them in:

```
let accountManager = VSUserAccountManager.shared
let autoSignInToken = try? await accountManager.autoSignInToken
if let token = autoSignInToken?.value {
    // Consume the token to bypass the sign-in screen, if appropriate.
}
```

Important

Only use an Automatic Sign-In token to enable media streaming, and provide access to streaming features, like watchlists.

Consider other mechanisms to secure a person's account itself, or to enable the app to display their personal information.

Manage the token on device or from a web server

You might revoke an Automatic Sign-In token if:

- Your app provides its own UI that lets the person opt out of Automatic Sign-In.
- The person changes their password and wants to sign out from all of their devices.

- Your app implements conditions to invalidate a token, such as if a person flags a specific log in as unauthorized.

To revoke an Automatic Sign-In token, call `deleteAutoSignInToken()`, and the framework sets the token:

- `value` to `nil`
- `authorization` to `VSUserAccountManager.AutoSignInAuthorization.notDetermined`.

The Automatic Sign-In API enables you to manage tokens on your web sever. For example, you might allow the person to opt out of Automatic Sign-In from your website.

To remove the current token value from your web server, call the `Delete Sign-In Token` endpoint:

```
https://api.storekit.itunes.apple.com/account/v1/autoSignIn/delete
```

In addition, you can update the current token value from your web server by calling `Update Sign-In Token`:

```
https://api.storekit.itunes.apple.com/account/v1/autoSignIn/update
```

See Also

User account management

`class` VSUserAccountManager

The object that coordinates your app's user account actions.

`struct` VSUserAccount

An object that represents a user's account.