

[Technotes](#) / TN3185: Troubleshooting In-App Purchases availability in Xcode

Article

TN3185: Troubleshooting In-App Purchases availability in Xcode

Inspect your active StoreKit configuration file for unexpected configurations.

Overview

When using [StoreKit Testing in Xcode](#) to test your In-App Purchases, your app may not display its products. StoreKit Testing in Xcode is a local test environment for testing In-App Purchases without requiring a connection to App Store servers. To set up testing in this environment, add a local or synced StoreKit configuration file that contains In-App Purchases to your Xcode project. For more information, see [Create a StoreKit configuration file](#). When you [enable a configuration file](#) in your Xcode project, this file becomes active. StoreKit uses data saved in the active configuration file when your app calls StoreKit APIs in the test environment. For more information, see [Setting up StoreKit Testing in Xcode](#).

To offer In-App Purchases in your app, call `Product.products(for:)` with a list of [product identifiers](#) (Product ID) matching these products in the test environment. `Product.products(for:)` returns an array that includes an instance of [Product](#) for each of the In-App Purchases. Update your app's UI with these returned instances, which contain all In-App Purchase information set up in the active configuration file for your app.

If `Product.products(for:)` fails to return a Product instance for your In-App Purchases, it may be due to the following reasons:

- Your In-App Purchases are missing or don't exist in the active StoreKit configuration file.
- You set up the test environment to simulate a load products failure scenario.

Note

If your app fails to display its products when testing In-App Purchases in the Apple sandbox environment, or when launching the app in the App Store, see [TN3186: Troubleshooting In-App Purchases availability in the sandbox](#) and [TN3188: Troubleshooting In-App Purchases availability in the App Store](#), respectively.

Validate your product identifier list

Inspect the active StoreKit configuration file in your Xcode project. Confirm each product identifier in your list matches the product identifier of an In-App Purchase configured in this file.

Disable the Simulated StoreKit Load Products failure setting

A [StoreKit configuration file](#) includes settings you can use to specify test conditions or scenarios for your In-App Purchases such as Load Products. When you enable Load Products, the test environment simulates the load product failure scenario you specified such as network error. Calling `Product.products(for:)` in your app throws a [StoreKit error](#). When you disable this setting, the function returns all your In-App Purchases that exist in the active configuration file. For more information, see [Change settings and initiate test conditions](#) in [Testing in-app purchases with StoreKit transaction manager in Xcode](#).

To prevent the test environment from simulating a load products failure scenario, perform these steps in your Xcode project:












1. In the Project navigator, select your active StoreKit configuration file.
2. Click Configuration Settings.
3. Scroll down to Simulated StoreKit Failures.
4. Disable the Load Products setting.

Revision History

- 2025-04-29 First published.

See Also

Latest

-  **TN3190: USB audio device design considerations**
Learn the best techniques for designing devices that conform to the USB Audio Device Class specifications.
-  **TN3194: Handling account deletions and revoking tokens for Sign in with Apple**
Learn the best techniques for managing Sign in with Apple user sessions and responding to account deletion requests.
-  **TN3193: Managing the on-device foundation model's context window**
Learn how to budget for the context window limit of Apple's on-device foundation model and handle the error when reaching the limit.
-  **TN3115: Bluetooth State Restoration app relaunch rules**
Learn about the conditions under which an iOS app will be relaunched by Bluetooth State Restoration.
-  **TN3192: Migrating your iPad app from the deprecated UIRequiresFullScreen key**
Support iPad multitasking and dynamic resizing while updating your app to remove the deprecated full-screen compatibility mode.
-  **TN3151: Choosing the right networking API**
Learn which networking API is best for you.
-  **TN3111: iOS Wi-Fi API overview**
Explore the various Wi-Fi APIs available on iOS and their expected use cases.
-  **TN3191: IMAP extensions supported by Mail for iOS, iPadOS, and visionOS**
Learn which extensions to the RFC 3501 IMAP protocol are supported by Mail for iOS, iPadOS, and visionOS.
-  **TN3134: Network Extension provider deployment**
Explore the platforms, packaging, OS versions, and device configurations for Network Extension provider deployment.
-  **TN3179: Understanding local network privacy**
Learn how local network privacy affects your software.
-  **TN3189: Managing Mail background traffic load**
Identify iOS Mail background traffic and manage its impact on your IMAP server.

 TN3187: Migrating to the UIKit scene-based life cycle

Update your app to receive scene-based life-cycle events and manage your user interface using scene objects and methods.

 TN3188: Troubleshooting In-App Purchases availability in the App Store

Verify your In-App Purchases are approved and available for sale in the App Store.

 TN3186: Troubleshooting In-App Purchases availability in the sandbox

Identify common configurations that make your In-App Purchases unavailable in the sandbox environment.

 TN3182: Adding privacy tracking keys to your privacy manifest

Declare the tracking domains you use in your app or third-party SDK in a privacy manifest.