Article

Setting up StoreKit Testing in Xcode

Prepare your test environment to test in-app purchases with data you configure locally.

Overview

StoreKit Testing in Xcode is a local test environment for testing in-app purchases without requiring a connection to App Store servers. Set up in-app purchases in a local StoreKit configuration file in your Xcode project, or create a synced StoreKit configuration file in Xcode from your in-app purchase settings in App Store Connect. After you enable the configuration file, the test environment uses this local data when your app calls StoreKit APIs.

> **Note**
>
> Enable Developer Mode to test your app on devices running iOS 16 and later, visionOS, or watchOS 9 and later. For more information about how to enable Developer Mode, see Enabling Developer Mode on a device.

As you test transactions, the test environment displays a payment sheet with localized values, and produces receipts for you to verify.

Testing in-app purchase scenarios with StoreKit in Xcode is useful for:

- Developing features that use in-app purchases before configuring them in App Store Connect.
- Testing locally when a network connection isn’t available.
- Debugging in-app purchase use cases that are harder to set up in the sandbox environment, such as eligibility for promotional offers.
- Viewing localized product information in the payment sheet.

- Testing transactions end-to-end, including failed transactions.

The full functionality of StoreKit Testing in Xcode is available for automation. See StoreKit Test for information about automating in-app purchase testing. For more information about testing StoreKit at different stages of development, see Testing at all stages of development with Xcode and the sandbox.

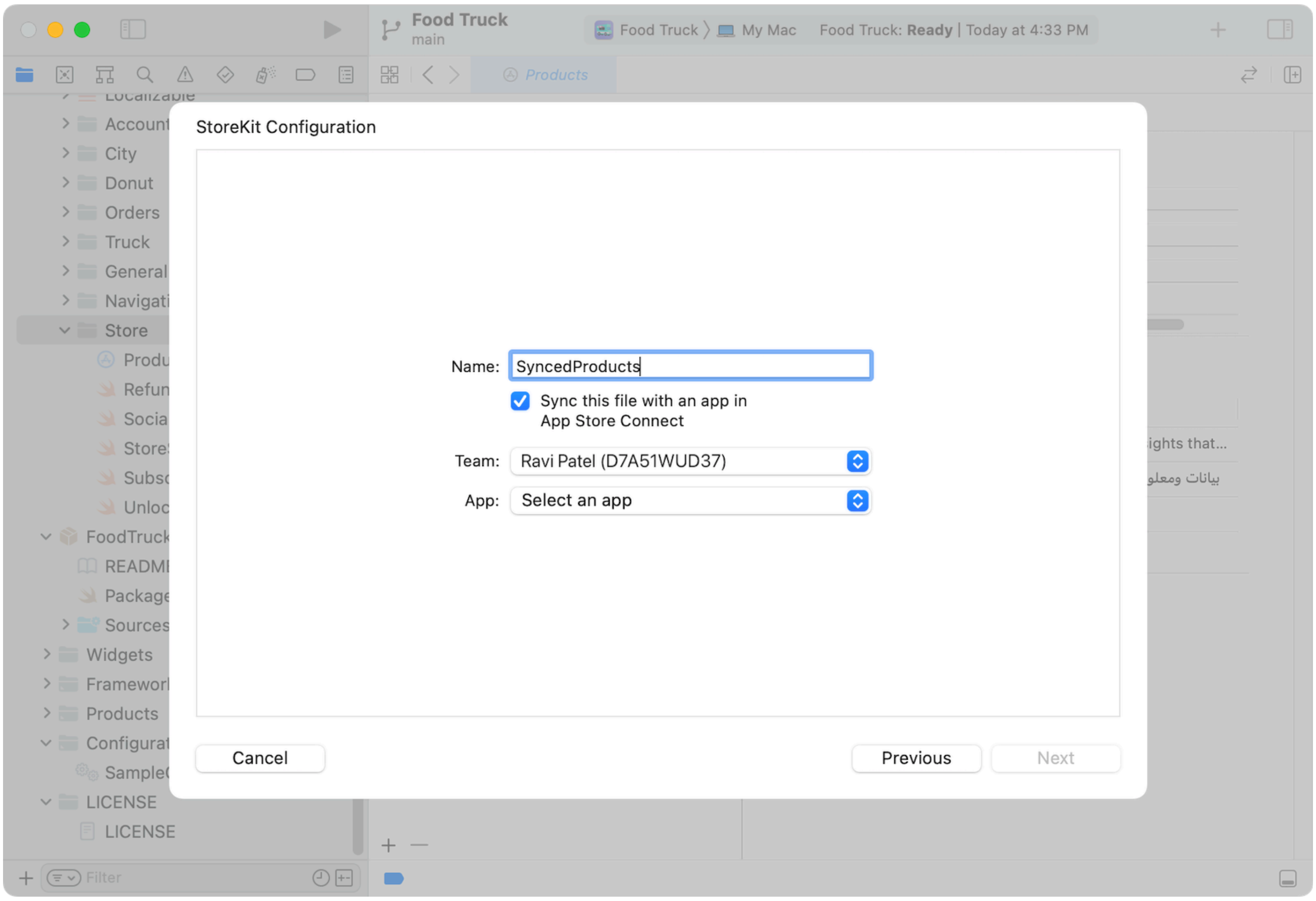
Create a StoreKit configuration file

A StoreKit configuration file contains descriptions of in-app purchases, subscription groups, auto-renewable subscriptions, and non-renewing subscriptions. When the configuration file is active, StoreKit uses this data when your app calls StoreKit APIs in the test environment. There are two types of StoreKit configuration files: local and, in Xcode 14 and later, synced.

- Set up a local configuration file if you haven't set up your app in App Store Connect, or you want to try out new types of in-app purchases or subscriptions before you set them up in App Store Connect. The local data is convenient to edit in Xcode, and stands in for the data that otherwise comes from App Store Connect.
- Set up a synced configuration file if you have in-app purchases or subscriptions already set up in App Store Connect that you want to test in Xcode.

To create a StoreKit configuration file:

1. Launch Xcode, then choose File > New > File From Template.
2. In the sheet that appears, enter *storekit* in the Filter search field.
3. Select StoreKit Configuration File, then click Next.
4. In the dialog, enter a name for the file. For a synced configuration file, select the checkbox, specify your team and app in the drop-down menus that appear, then click Next. For a local configuration, leave the checkbox unselected, then click Next.
5. Select a location, and click Create.
6. Save the file to your project.


StoreKit Configuration

Name: SyncedProducts

☑ Sync this file with an app in App Store Connect

Team: Ravi Patel (D7A51WUD37)

App: Select an app

If you rename the configuration file, be sure to keep its file extension `.storekit`.

Note

You can convert a synced configuration file to a local configuration file. If you want to sync again, create a new synced configuration file.

Set up a StoreKit configuration

To edit the settings in a local StoreKit configuration file, select the file in the Project navigator to open the custom editor. Click the Add button (+) in the editor to add product details to the configuration file.

Note

You can’t edit a synced configuration file unless you convert it to be a local configuration file. To convert a synced configuration file, select the file, then choose Editor > Convert to Local StoreKit Configuration from the Xcode menu. Click Convert File in the confirmation dialog. When you’re viewing a synced configuration file, click the Sync button in the bottom left corner to pull the latest updates from App Store Connect.

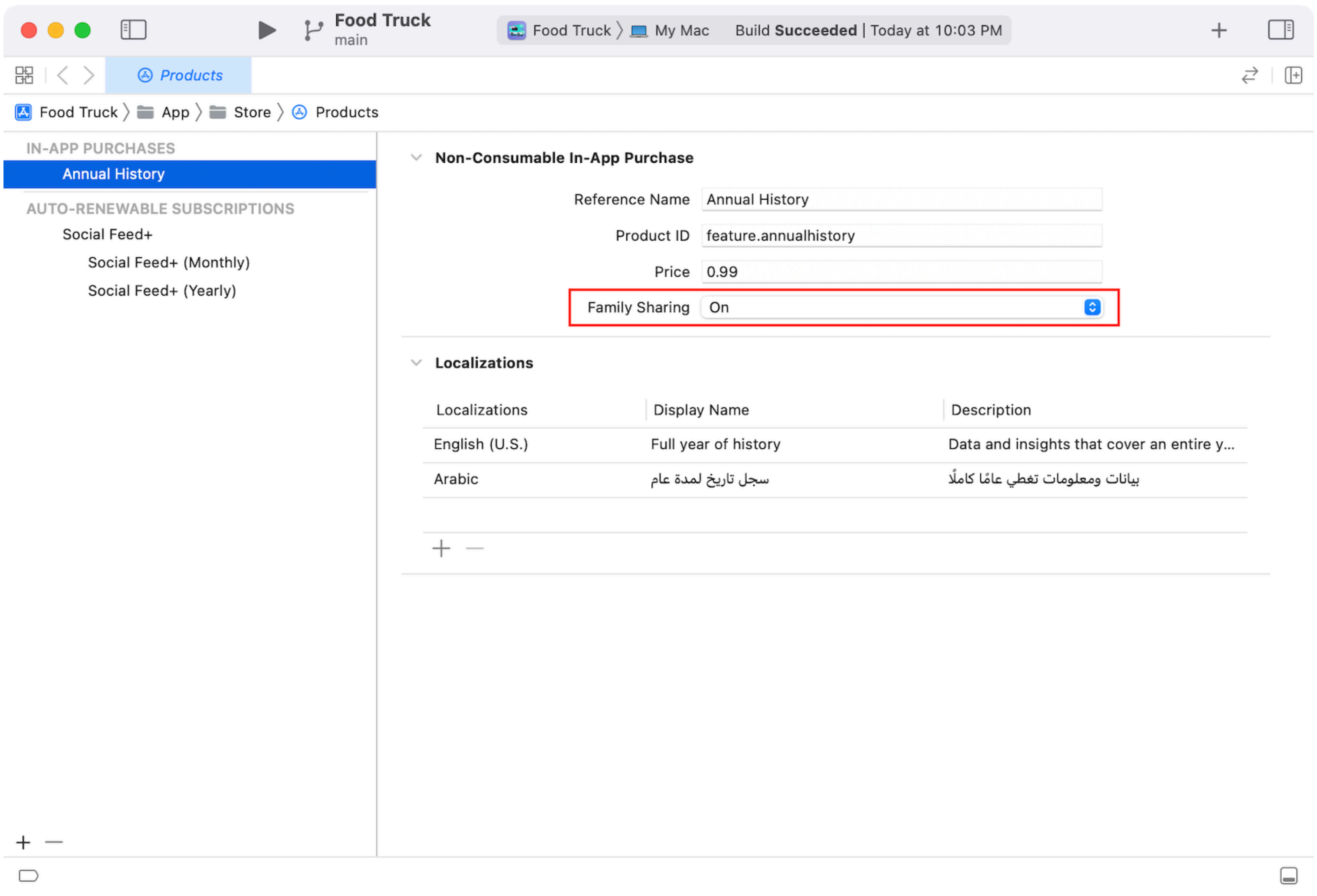
Enter the information in your local StoreKit configuration file manually. The product names, IDs, prices, localizations, and any other data you provide in the StoreKit configuration file don’t upload to App Store Connect, and don’t appear in App Store-signed apps. App Store Connect data transfers only to a synced StoreKit configuration file, which you can’t edit in Xcode.

Tip

If you have a synced configuration file with items you want to test in a local configuration file, you may copy an item from your synced configuration file to your local configuration file. To do this, Control-click an item in your synced configuration file, and choose Copy. In your local configuration file, select where you want to place the item, and choose Edit > Paste from the Xcode menu.

Each in-app purchase and non-renewing subscription has a reference name, product ID, and price. In-app purchases and non-renewing subscriptions optionally have localizations. This metadata shows up in the payment sheet after you click to buy a product in your app. Note that the price is a placeholder value that’s not connected to price tiers or real pricing information. However, it appears using the correct currency symbols for the storefront you’re testing.

For non-consumable in-app purchases and auto-renewable subscriptions, select Family Sharing to mark the product for sharing as the following image shows:



Non-Consumable In-App Purchase

Reference Name	Annual History
Product ID	feature.annualhistory
Price	0.99
Family Sharing	On

Localizations

Localizations	Display Name	Description
English (U.S.)	Full year of history	Data and insights that cover an entire y...
Arabic	سجل تاريخ لمدة عام	بيانات ومعلومات تغطي عامًا كاملًا

When you set up the first auto-renewable subscription, Xcode prompts you to create the first subscription group. After adding subscriptions to a group, the level you assign to each subscription determines their upgrade and downgrade options. For more information, see Offer auto-renewable subscriptions.

For auto-renewable subscriptions, set up introductory offers and promotional offers with the options available in App Store Connect. To begin testing, configure at least one in-app purchase product.

Enable StoreKit Testing in Xcode

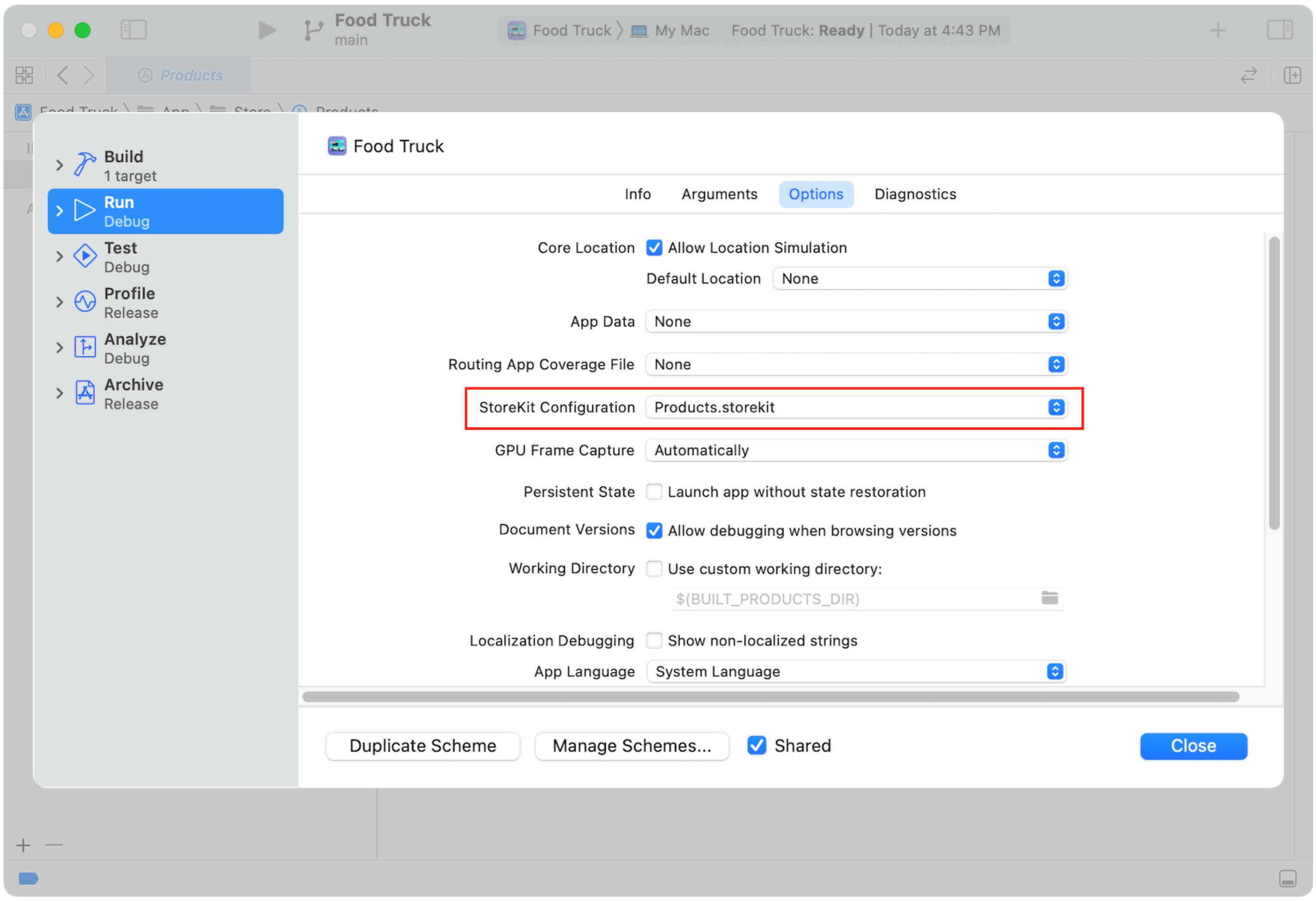
To enable StoreKit Testing in Xcode, your project must have an active StoreKit configuration file. By default, StoreKit Testing in Xcode is disabled. To select a configuration file and make it active:

1. Click the scheme to open the Scheme menu and choose Edit Scheme.

2. In the scheme editor, select the Run action.

3. Click the Options tab.

4. For the StoreKit Configuration option, select a configuration file and click Close.



Core Location: ☑ Allow Location Simulation
Default Location: None
App Data: None
Routing App Coverage File: None
StoreKit Configuration: Products.storekit
GPU Frame Capture: Automatically
Persistent State: ☐ Launch app without state restoration
Document Versions: ☑ Allow debugging when browsing versions
Working Directory: ☐ Use custom working directory:
$(BUILT_PRODUCTS_DIR)
Localization Debugging: ☐ Show non-localized strings
App Language: System Language
☑ Shared

You can also add an existing StoreKit configuration file to the project from this menu. Choose a configuration file with a `.storekit` file extension.

An Xcode project can contain multiple StoreKit configuration files, but only one can be active at a time. When it's active, build and run your app as usual. Instead of accessing App Store Connect or the sandbox server, your app gets StoreKit data from the test environment.

Disable StoreKit Testing in Xcode

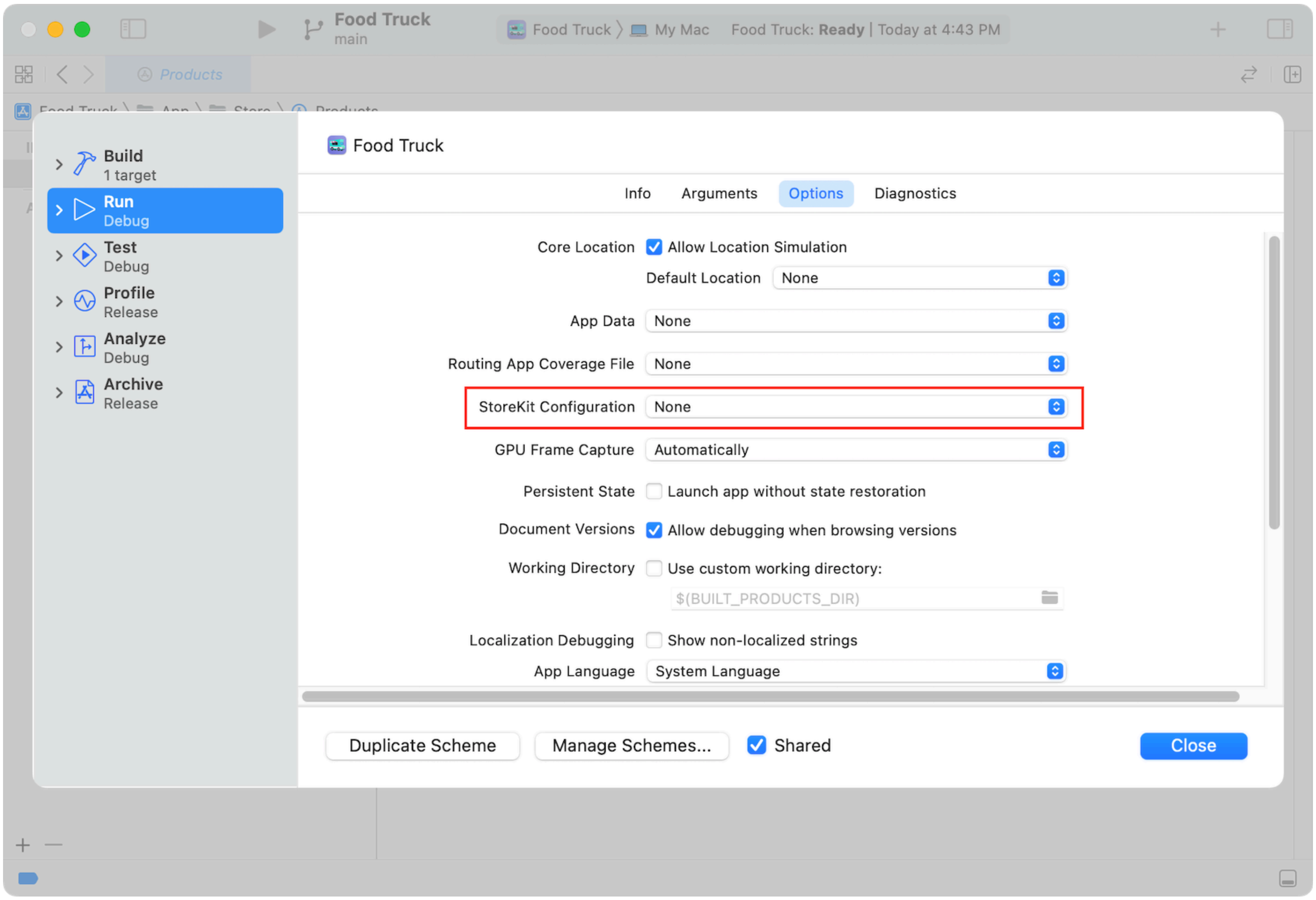
To disable StoreKit Testing in Xcode, remove the StoreKit configuration file from the scheme's run options:

1. Click the scheme to open the Scheme menu and choose Edit Scheme.

2. In the scheme editor, select the Run action.

3. Click the Options tab.

4. For the StoreKit Configuration option, select None.



Core Location ✓ Allow Location Simulation
Default Location: None
App Data: None
Routing App Coverage File: None
StoreKit Configuration: None
GPU Frame Capture: Automatically
Persistent State ☐ Launch app without state restoration
Document Versions ✓ Allow debugging when browsing versions
Working Directory ☐ Use custom working directory:
$(BUILT_PRODUCTS_DIR)
Localization Debugging ☐ Show non-localized strings
App Language: System Language

Your app stops using the local data from the configuration file and starts using the data from App Store Connect. For more information, see Overview for configuring in-app purchases.

Prepare to validate receipts in the test environment

StoreKit Testing in Xcode generates locally signed receipts that your app validates locally. Obtain the certificate you need for local validation and add it to your project as follows:

1. In Xcode's Project navigator, click the StoreKit configuration file.
2. From the Xcode menu, choose Editor > Save Public Certificate.
3. Select a location in your project to save the file.

> **Note**
>
> The test environment's certificate is a root certificate. There's no certificate chain to validate when you validate the receipt signature.

Be sure your code uses the correct certificate in all environments. Add the following conditional compilation block to your receipt validation code to select the test certificate for testing, and the Apple root certificate, otherwise:

```
#if DEBUG
    let certificate = "StoreKitTestCertificate"
#else
    let certificate = "AppleIncRootCertificate"
#endif
```

Your code is ready to validate receipts by selecting the appropriate certificate in the test environment and in the production environment.

Important

Receipts you produce in the test environment aren’t signed by the App Store and aren’t valid for apps in production.

See Also

StoreKit

Testing in-app purchases with StoreKit transaction manager in Xcode

Use the transaction manager within Xcode to test in-app purchases without requiring a connection to App Store servers.