App Thinning (iOS, tvOS, watchOS)

The store and operating system optimize the installation of iOS, tvOS, and watchOS apps by tailoring app delivery to the capabilities of the user's particular device, with minimal footprint. This optimization, called *app thinning*, lets you create apps that use the most device features, occupy minimum disk space, and accommodate future updates that can be applied by Apple. Faster downloads and more space for other apps and content provides a better user experience.

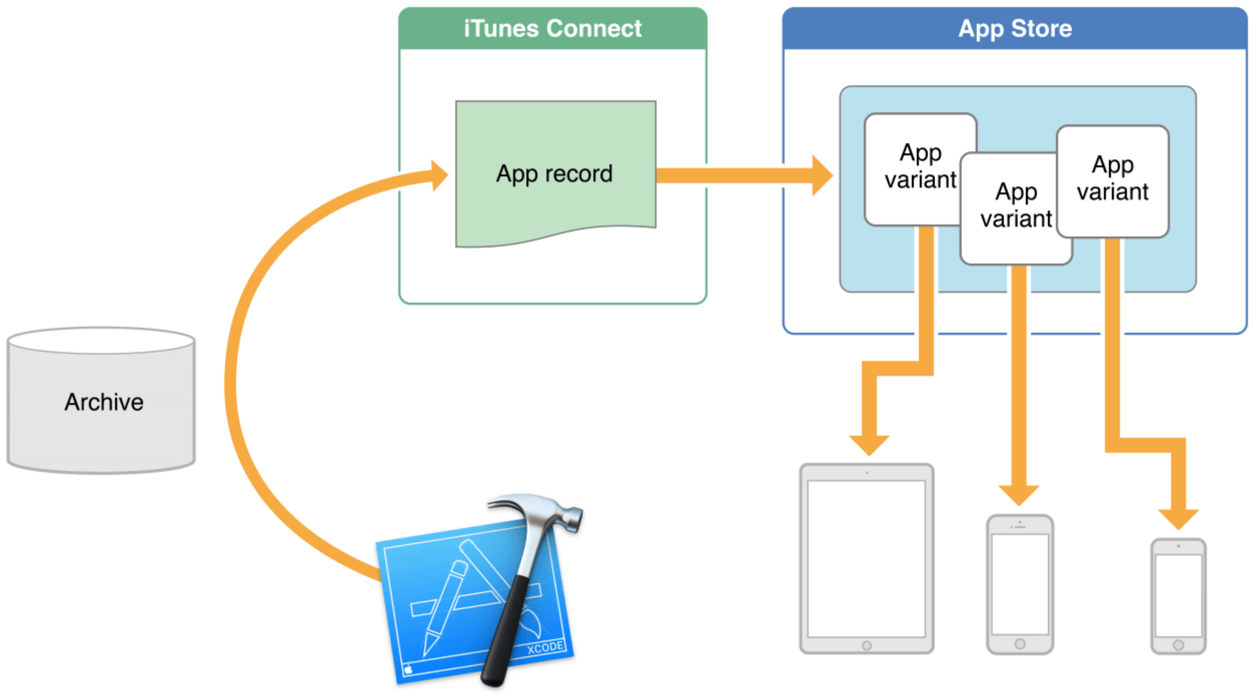
This chapter describes the three components of app thinning: slicing, bitcode, and on-demand resources.

Slicing (iOS, tvOS)

Slicing is the process of creating and delivering variants of the app bundle for different target devices. A *variant* contains only the executable architecture and resources that are needed for the target device. You continue to develop and upload full versions of your app to iTunes Connect. The store will create and deliver different variants based on the devices your app supports. Image resources are sliced according to their resolution and device family. GPU resources are sliced according to device capabilities. For tvOS apps, assets in catalogs shared between iOS and tvOS targets are sliced and large app icons are removed. When the user installs an app, a variant for the user's device is downloaded and installed.

Xcode simulates slicing during development so you can create and test variants locally. Xcode slices your app when you build and run your app on a device. When you create an archive, Xcode includes the full version of your app but allows you to export variants from the archive.

Note: Sliced apps are supported on devices running 9.0 and later; otherwise, the store delivers universal apps to customers.



Slicing is performed during your normal development and distribution workflows, which proceed generally as follows:

1. In Xcode, specify target devices and provide multiple resolutions of images in the asset catalog.

   You must use the asset catalog in order for resources to be sliced.

2. Build and run the app in Simulator or on a device.

   Xcode builds a variant of the app for the selected device type, improving the debug build time and allowing you to test variants locally.

3. Create an archive of the app and export a variant locally for target devices.

   Test all the variants you export on target devices to discover configuration issues early.

4. Upload the app to iTunes Connect.

   The store creates individual app variants from the archive. The number of variants depends on the architectures and resources specified in the Xcode project.

5. In iTunes Connect, distribute a prerelease version of your app to designated testers.

   Testers install the prerelease version on supported devices using TestFlight. TestFlight downloads a variant of the app specific to the user's device.

   > **Note:** To test the variants that the store builds before you distribute your app to users, invite internal testers (your team's iTunes Connect users) only and download the variants using TestFlight. If you invite external testers (users specifying only their email addresses), the external testers must wait for Beta App Review to approve the app before they can download variants.

6. In iTunes Connect, release the app.

   Users install the app on supported devices, and the store app downloads a variant of the app specific to the user's device.

Bitcode

Bitcode is an intermediate representation of a compiled program. Apps you upload to iTunes Connect that contain bitcode will be compiled and linked on the store. Including bitcode will allow Apple to re-optimize your app binary in the future without the need to submit a new version of your app to the store.

Xcode hides symbols generated during build time by default, so they are not readable by Apple. Only if you choose to include symbols when uploading your app to iTunes Connect would the symbols be sent to Apple. You must include symbols to receive crash reports from Apple.

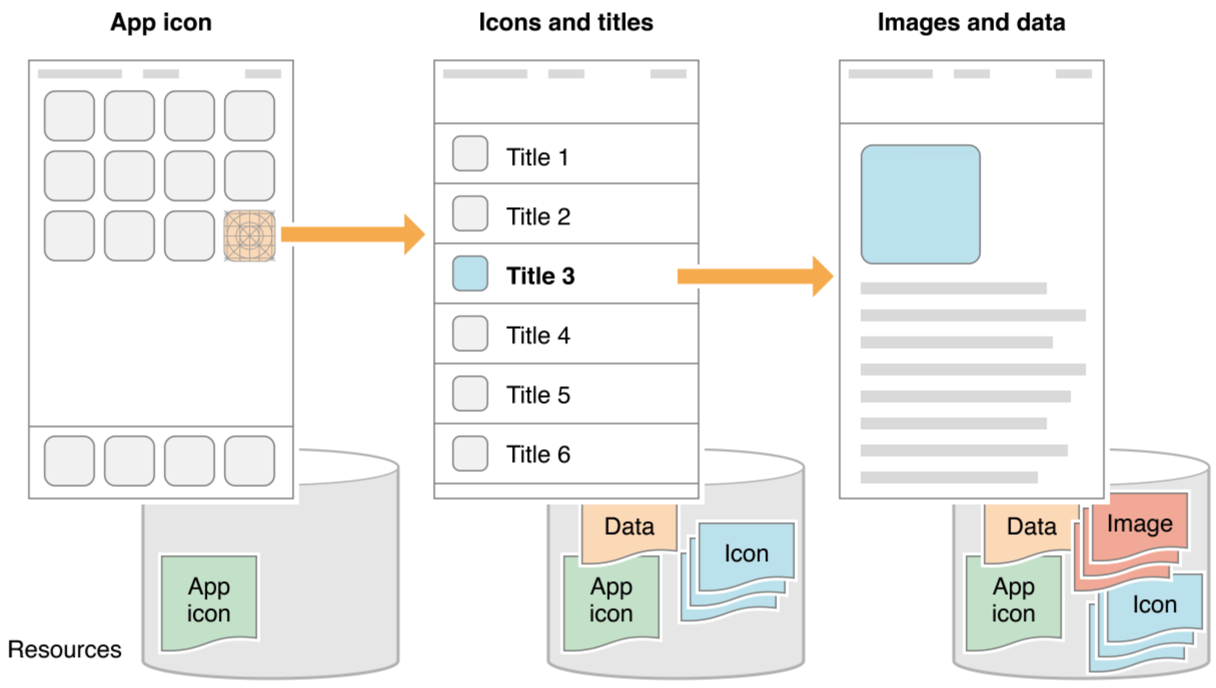
> **Note:** For iOS apps, bitcode is the default, but optional. For watchOS and tvOS apps, bitcode is required. If you provide bitcode, all apps and frameworks in the app bundle (all targets in the project) need to include bitcode.
>
> After you distribute your app using iTunes Connect, you can download the dSYMs file for the build, described in Viewing and Importing Crashes in the Devices Window.

On-Demand Resources (iOS, tvOS)

On-demand resources are resources—such as images and sounds—that you can tag with keywords and request in groups, by tag. The store hosts the resources on Apple servers and manages the downloads for you. On-demand resources enable faster downloads and smaller app sizes, improving the first-time launch experience. For example, a game app may divide resources into game levels and request the next level of resources only when the app anticipates that the user will move to that level. Similarly, the app can request In-App Purchase resources only when the user buys the corresponding in-app purchase.

The operating system purges on-demand resources when they are no longer needed and disk space is low. If you export your app for testing or distribution outside of the store, you must host the on-demand resources yourself. Note that executable on-demand resources are not supported. The store also slices on-demand resources, as described in Slicing (iOS, tvOS), further improving the user experience.

For users, on-demand resources work transparently in the background, supplying resources as needed while the user explores the features of your app.



To set up on-demand resources in your app, read *On-Demand Resources Guide* and *NSBundleResourceRequest Class Reference*.