Designing for App Sandbox

There's a common, basic workflow for designing or converting an app for App Sandbox. The specific steps to take for your particular app, however, are as unique as your app. To create a work plan for adopting App Sandbox, use the process outlined here, along with the conceptual understanding you have from the earlier chapters in this document.

Six Steps for Adopting App Sandbox

The workflow to convert a Mac OS X app to work in a sandbox typically consists of the following six steps:

1. Determine whether your app is suitable for sandboxing.
2. Design a development and distribution strategy.
3. Resolve API incompatibilities.
4. Apply the App Sandbox entitlements you need.
5. Add privilege separation using XPC.
6. Implement a migration strategy.

Determine Whether Your App Is Suitable for Sandboxing

Most Mac OS X apps are fully compatible with App Sandbox. If you need behavior in your app that App Sandbox does not allow, consider an alternative approach. For example, if your app depends on hard-coded paths to locations in the user's home directory, consider the advantages of using Cocoa and Core Foundation path-finding APIs, which use the sandbox container instead.

If you choose to not sandbox your app now, or if you determine that you need a temporary exception entitlement, use Apple's bug reporting system to let Apple know what's not working for you. Apple considers feature requests as it develops the Mac OS X platform.

The following app behaviors are *incompatible* with App Sandbox:

- Use of Authorization Services

  With App Sandbox, you cannot do work with the functions described in *Authorization Services C Reference*.

- Use of accessibility APIs in assistive apps

  With App Sandbox, you can and should enable your app for accessibility, as described in *Accessibility Overview*. However, you cannot sandbox an assistive app such as a screen reader, and you cannot sandbox an app that controls another app.

- Sending Apple events to arbitrary apps

  With App Sandbox, you can receive Apple events and respond to Apple events, but you cannot send Apple events to arbitrary apps.

  By using a temporary exception entitlement, you can enable the sending of Apple events to a list of specific apps that you specify, as described in *Entitlement Key Reference*.

- Sending user-info dictionaries in broadcast notifications

  With App Sandbox, you can send notifications, but you cannot include a user-info dictionary in the notifications you send.

- Loading kernel extensions

  Loading of kernel extensions is prohibited with App Sandbox.

- Simulation of user input in Open and Save dialogs

  If your app depends on programmatically manipulating Open or Save dialogs to simulate or alter user input, your app is unsuitable for sandboxing.

- Setting preferences on other apps

  With App Sandbox, each app maintains its preferences inside its container. Your app has no access to the preferences of other apps.

- Terminating other apps

  With App Sandbox, you cannot use the `NSRunningApplication` class to terminate other apps.

Design a Development and Distribution Strategy

During development, you may have occasion to run versions of your app that are signed with different code signatures. After you've run your app signed using one signature, the system won't allow a second version of your app, signed with a second signature, to launch—unless you modify the app's container. Be sure to understand how to handle this, as described in "App Sandbox and Code Signing," as you design your development strategy.

When a customer first launches a sandboxed version of your app, the system creates a container for your app. The access control (ACL) list for the container is established at that time, and the ACL is tied to the code signature of that version of your app. The implication for you is that all future versions of the app that you distribute must use the same code signature.

To learn how to obtain code signing certificates from Apple, read "Creating Signing Certificates" in *Mac App Development Workflow Guide*.

Resolve API Incompatibilities

If you are using Mac OS X APIs in ways that were not intended, or in ways that expose user data to attack, you may encounter incompatibilities with App Sandbox. This section provides some examples of app design that are incompatible with App Sandbox and suggests what you can do instead.

Opening, Saving, and Tracking Documents

If you are managing documents using any technology other than the `NSDocument` class, you must convert to using this class. The `NSDocument` class automatically works with Powerbox. `NSDocument` also provides support for keeping documents within your sandbox if the user moves them using Finder.

Creating a Login Item for Your App

With App Sandbox, you cannot create a login item using functions in the `LSSharedFileList.h` header file. For example, you cannot use the function `LSSharedFileListInsertItemURL`.

Instead, use the `SMLoginItemSetEnabled` function along with the `LSRegisterURL` function, as described in "Adding Login Items Using the Service Management Framework" in *Daemons and Services Programming Guide*.

Accessing User Data

Mac OS X path-finding APIs, above the POSIX layer, return paths relative to the container instead of relative to the user's home directory. If your app, before you sandbox it, accesses locations in the user's actual home directory (~) and you are using Cocoa or Core Foundation APIs, then, after you enable sandboxing, your path-finding code automatically uses your app's container instead.

For first launch of your sandboxed app, Mac OS X automatically migrates your app's main preferences file. If your app uses additional support files, perform a one-time migration of those files to the container, as described in

"Migrating an App to a Sandbox."

If you are using a POSIX command such as `getpwuid` to obtain the path to the user's actual home directory, consider instead using a Cocoa or Core Foundation symbol such as the `NSHomeDirectory` function. By using Cocoa or Core Foundation, you support the App Sandbox restriction against directly accessing the user's home directory.

If your app requires access to the user's home directory in order to function, let Apple know about your needs using the Apple bug reporting system. In addition, be sure to follow the guidance regarding entitlements provided on the iTunes Connect website.

Accessing Preferences of Other Apps

Because App Sandbox directs path-finding APIs to the container for your app, reading or writing to the user's preferences takes place within the container. Preferences for other sandboxed apps are inaccessible. Preferences for apps that are not sandboxed are placed in the `~/Library/Preferences` directory, which is also inaccessible to your sandboxed app.

If your app requires access to another app's preferences in order to function—for example, if it requires access to the playlists that a user has defined for iTunes—let Apple know about your needs using the Apple bug reporting system. In addition, be sure to follow the guidance regarding entitlements provided on the iTunes Connect website.

With these provisos in mind, you can use a path-based temporary exception entitlement to gain programmatic access to the user's `~/Library/Preferences` folder. Use a read-only entitlement to avoid opening the user's preferences to malicious exploitation. A POSIX function, such as `getpwuid`, can provide the file system path you need. For details on entitlements, see *Entitlement Key Reference*.

Apply the App Sandbox Entitlements You Need

To adopt App Sandbox for a target in an Xcode project, apply the `<true/>` value to the `com.apple.security.app-sandbox entitlement` key for that target. Do this in the Xcode target editor by selecting the Enable App Sandboxing checkbox.

Apply other entitlements as needed. For a complete list, refer to *Entitlement Key Reference*.

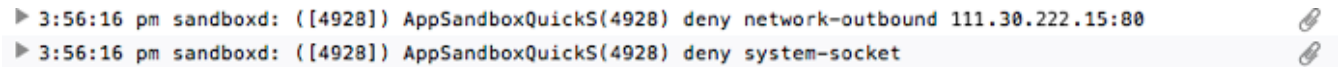
> **Important:** App Sandbox protects user data most effectively when you minimize the entitlements you request. Take care not to request entitlements for privileges your app does not need. Consider whether making a change in your app could eliminate the need for an entitlement.

Here's a basic workflow to use to determine which entitlements you need:

1. Run your app and exercise its features.
2. In the Console app (available in `/Applications/Utilities/`), look for `sandboxd` violations in the All Messages system log query.

   Each such violation indicates that your app attempted to do something not allowed by your sandbox.

   Here's what a `sandboxd` violation looks like in Console:

   Click the paperclip icon to the right of a violation message to view the backtrace that shows what led to the violation.
3. For each `sandboxd` violation you find, determine how to resolve the problem. In same cases, a simple change to your app, such as using your Container instead of other file system locations, solves the problem. In other cases, applying an App Sandbox entitlement using the Xcode target editor is the best choice..
4. Using the Xcode target editor, enable the entitlement that you think will resolve the violation.
5. Run the app and exercise its features again.

Either confirm that you have resolved the `sandboxd` violation, or investigate further.

If you choose not to sandbox your app now or to use a temporary exception entitlement, use Apple's bug reporting system to let Apple know about the issue you are encountering. Apple considers feature requests as it develops the Mac OS X platform.

Add Privilege Separation Using XPC

When developing for App Sandbox, look at your app's behaviors in terms of privileges and access. Consider the potential benefits to security and robustness of separating high-risk operations into their own XPC services.

When you determine that a feature should be placed into an XPC service, do so by referring to "Creating XPC Services".

Implement a Migration Strategy

Ensure that customers who are currently using a pre-sandbox version of your app experience a painless upgrade when they install the sandboxed version. For details on how to implement a container migration manifest, read "Migrating an App to a Sandbox."