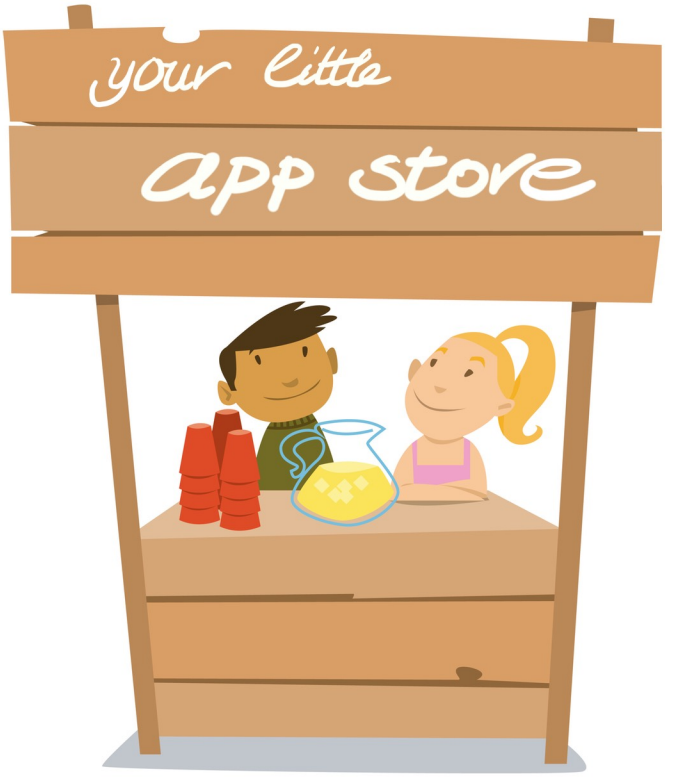
v1.1.0

Swift 3

featuring FastSpring



Make Money

Outside the Mac App Store

Christian Tietze

Make Money Outside the Mac App Store

How to Sell Your Mac App with FastSpring, Secure It With License Codes Against Piracy, and Offer Time-Based Trial Downloads

Christian Tietze

This book is for sale at
http://leanpub.com/sell-mac-app-fastspring-cocoafob-license-trial

This version was published on 2017-02-14

To my Mastermind Deacon, Rob, and Tristan: I owe you for your support and inspiration.

Contents

We appreciate Christian's efforts in creating a guide that enables Mac developers to sell applications through FastSpring's award-winning e-commerce platform. He has provided detailed instructions to help developers configure key elements of their online sales process. The spirit of community captured in his book reflects FastSpring's mission to connect people globally in the digital economy.

— Mike Smith, CTO FastSpring, December 2015

What This Book is About

So you consider releasing a Mac app outside the Mac App Store.

Fantastic!

I think that's cool, because:

- You will earn more per sale – most store fronts on the market will take a cut between 6 and 10% only, as opposed to the 30% Apple takes *after taxes*.[1] Two companies left the App Store in 2016 and shared their numbers. They report a slight decline in units sold but a growth in total revenue because of Apple's absurd 30% commission.[2]
- You will not be hampered by App Store Sandboxing restrictions: even though Sandboxing is said to be the way to go for the future, releasing a Sandboxed app in the App Store will impose even stricter limitations, like not having access to assistive devices, observing keyboard inputs, or registering audio drivers.
- You own the update process: you can publish updates more often and react to bugs quicker. When you release an update, it's live immediately.
- You are not affected by paid search ads. The iOS App Store ranks paid search ads for keywords higher than apps of the same name. Imagine searching for "Uber" and the first result is the competitor, Lyft, and vice-versa. I understand that Google sells ad space in search result pages, but your affiliation with Apple is different and *should* be treated differently.
- You can add copy protection to you app. No security measure is 100% perfect and it doesn't pay off to invest too much time in securing your application against cracking, but still: stuff you buy on the Mac App Store is usually not even secured against copying with Apple-ID authorization.[3]

[1] Say you sell an app for €9.99 to a German customer. After a 19% VAT reduction of €1.90, Apple takes 30% of the remaining €8.09 and you'll earn a depressing €5.66 – whereas the same app sold via FastSpring would leave you with €8.14 after the 8.9% order fee.

[2] Rogue Amoeba published "Making More Outside The App Store" in February 2017; Kapeli published 100 Days Without the App Store in late January 2017.

[3] http://www.macworld.com/article/1157018/appstore_licensing.html

- You can issue discounts: student discounts, bulk sale discounts, non-profit discounts, time-limited price drops, bundle sales – you name it.
- You will know your customers and get in touch with them. This saved my back in the past when my app wouldn't update automatically and I wanted to tell my customers how to get the latest version. Same holds true for special offers or upselling existing customers.

Speaking of discounts: you can also issue what **Craig Scott of toketaWare** calls "good will discounts." There are people who'll pay for your product as long as they think it's a good deal. If there are people who are angry because they think your product is too expensive: give them a few percent off to convert them to paying customers.

The surprise can be very effective: "Quite often your angriest customer can be 'turned' into your most vociferous advocate. I don't like 'giving grease to the squeaky wheel' but it does work!" This is a great lesson: don't be pissed if people treat you harshly. Surprise them with a small token of appreciation and their negative emotions will change for the better and create a strong connection.

Same applies to those that seem "deserving," which is similar to the angry folks – only without the anger. It's a mindset of suffering when people approach you and say they're poor but still want to use the app. Since the app is your product, if you want to do someone a favor and offer a discount, then by all means go ahead and do it. They'll be thankful for it. You may even win their hearts and loyalty.

Selling on the Mac App Store offers benefits for you, too, which you shouldn't carelessly dismiss:

- Apple's App Stores are a very important marketplace where people discover new things. It's a place where some "go shopping." (Although I can hardly imagine how that works in practice.)
- Releasing an update equals pushing a binary to the iTunes Connect web service. After it passed the review process (which used to take a few days but is now done in about 1 day, according to http://appreviewtimes.com), people can

download it from one central place: the App Store application. When updates download automatically, users will always work with the latest version.

Philip Goward, **founder of Smile**, calls the Mac App Store a "warehouse." If you consider your application to be a commodity, a warehouse is the right place to put it.

If you build a great app targeted at professionals with "pro-pricing" around $100 or above, once you have build an audience for your product the oft-cited discoverability benefit of selling on the Mac App Store becomes miniscule. Having full control over sales and customer service then becomes more important.

Distributing via the App Store is just so convenient. Everyone can see your product and download and update from one central place. On the flip side, you lose money for each transaction, you're bound to the strict App Store security policies, and if Apple disables your account by mistake, you're screwed.

Not certain what to do now? Even though App Store-bashing seems to be quite popular, it's not obvious what developers really think and do. Are they all talk and no action? DevMate conducted a survey in 2016 about the Mac App Store and collected 679 entries. You can find the results online[4]. The survey isn't representative. Use this for motivation or to validate your existing opinion – but don't make rash decisions based on that. Try the process for yourself and see how *your* business is doing. Does distribution outside the Mac App Store help or hamper business? As always, your results may depend on many variables, so take careful steps.

When you distribute outside the Mac App Store, you have to take care of app updates and copy protection yourself, too. To figure all that out takes quite some time. I wrote this book to save you days of research, implementing and testing. This book is made to get your app out there quickly so you can learn what that means and truly be your own boss – be it on the Mac App Store or outside.

[4] http://devmate.com/mac-dev-survey

What You'll Learn

You'll learn how to perform the following tasks:

- Set up a product for sale with FastSpring, including license code generation and order confirmation e-mails.
- Guard your app against software piracy, requiring license code and name to let users pass.
- Implement a registration handler in your app which verifies the license information. *You can copy this well-tested code right into your existing app!*
- Add a time-based trial to your application. *This, too, can be copied right into your existing app!*

As a bonus, you'll learn a few things about Mac software architecture through well-written, fully functional sample applications which are thoroughly tested. There are more than 200 test cases included in the code repository!

I'll show you all these things in detail so you can be certain that everything is taken care of:

- how to set up FastSpring as the store where people buy your software,
- add automatic license key generation to the checkout process,
- implement license code verification in you app,
- create a time-based trial,
- offer in-app purchases,
- set up app updates,
- and how to figure out a price and prepare the release.

This book is designed to make your job an independent software developer as simple as possible. When I wrote this book, I had in mind how hard selling my first Mac app was, so I try to talk about the usual fears of the process. But if you're experienced with releasing apps already, you'll find ample advice on how to transition away from the Mac App Store.

After reading this book, you'll be able to quickly add license code verification to an existing app and lock out users who won't pay after the trial expires. You'll even be able to offer a store front from right within your application using FastSpring's *Embedded Store SDK* which may increase the conversion from demo users to paid customers. All of this is designed to work as a drop-in: you add the files to your project and don't have to worry about the details anymore.

Then there's the part about implementing time-based trials in your app. Time-based trials are the de facto standard to limit functionality of what was used to be called "shareware" in the '90s. Keeping track of time is another functionality you would have to add to your app but which I have taken care of. It integrates into the existing license mechanism nicely.

Next to time-based trials, it's also popular to offer feature-limited trials. There, you offer a demo version with limited capability from the start but potentially running forever. Users have to pay to unlock the full potential of the app. We won't cover that in much detail because it'll turn out that this is essentially an in-app purchase. Since you eill learn how to add in-app purchases to unlock features, you will be able to create feature-based trials, too.

All in all, you only have to take the well-factored sample code of this book and copy the parts you want to use into your awesome app to get started. If your app is already functional, it may take you no more than half an hour to put your app under copy protection and add a time-based trial – depending on your existing code, of course.

The sample code will be written in the latest stable version of Swift. I'll update the book regularly, so check back for a new e-book version download when you notice Swift has changed.

Why FastSpring?

I'm not at all affiliated with FastSpring. I love their service and got in touch when I was finishing the book – and they in turn loved this book. So we collaborate on spreading the word. Not because they pay me (they don't) but because I believe "making it" with Mac apps is possible, and FastSpring will be a good choice to get there.

A few years ago, I picked FastSpring to sell my apps mostly because of social proof. Lots of indie teams I respected for years already used FastSpring, so I figured their service was a good fit and gave it a spin.[5] I still love their service. That's why I wrote this book to help you become independent from Apple's stores.

Some obvious competitors that handle payments are:

- Paddle
- Avangate, e-sellerate, kagi
- gumroad
- PayPal and Stripe

PayPal and Stripe will handle purchases, and that's it. Figuring out VAT yourself is not fun and the time needed to make the store really do its job isn't worth the lower transaction fee.

Gumroad will also offer a simple store widget and lets you upload files. Credit cards work way better here than with PayPal, but not everyone around the world has a credit card in the first place – and I'm not talking about lesser developed countries: It's quite rare to know someone in Germany in 2015 with a credit card.

Kagi and e-sellerate are apt to create a real custom e-stores. So is FastSpring. All of these take care of handling VAT (value added tax) for you, which otherwise is a mess in the European Union.

If you want a more subjective argument: The great customer service sets FastSpring apart. For example, they offer you to take care of the initial store design *for free.* They are super responsive. A few people I interviewed pointed out they were happy with the quick and helpful responses – even though I didn't ask. That's my experience, too. So much for the anecdotes.

[5] Also, you can read about a small survey from 2009 where FastSpring scored quite high.

Pieter Omvlee revealed that **Bohemian Coding**, creators of the infamous Sketch design tool, initially sold via PayPal around 8 years ago. Bohemian Coding moved to FastSpring because of the better features, namely handling VAT automatically. Even though Sketch was sold exclusively on the Mac App Store when Apple opened its doors, after re-introducing FastSpring for volume licenses it turns out their own FastSpring-based store performs better than the Mac App Store.

Tyler Hall started his business by selling VirtualHostX via PayPal from 2007–2009, too. But he switched once a friend of his was locked out of PayPal due to "suspicious" activity. Imagine your sales going strong, then suddenly your payment processor blocks your account. That equals sudden death (or at least a severe and prolonged coma) for your business.

Account terminations still happen. There are many recorded incidents of people using amazon to sell books or being an affiliate. Then there was this weird story in late 2016 about an app named "Dash" which was removed from the Mac App Store when the developer's account was terminated. The point is: you can't do a thing when that happens to you.[6] Apple and amazon are huge, so false positive charges are inevitable. They also need to be proactive and exercise ownership to protect their customers – which is not you, the developer, but the end user. Selling on your own turns things around. To an e-commerce provider, you are the store owner and you are their customer. The end user is your responsibility, not theirs. That is a huge difference.

Keep in mind that I got in touch with a lot more of the people behind the curtains at FastSpring when I wrote this book than you might ever need to. During the spring 2017 webinar development it felt as if I was introduced to the whole family. I am getting more positively biased every year and work hard to be more helpful to you, dear reader, than a mere evangelist.

If you live outside the U.S., rest assured that FastSpring pays on time, twice a month, every month. I'm located in Germany, so I expected trouble when receiving

[6] In 2016, Dash, the popular documentation browser for Mac, was removed from the App Stores because the developer's acoount was cancelled. This can happen quickly and the lack of control from your side is threatening. (Read more on my website: https://christiantietze.de/posts/2016/10/dash-removed-from-app-stores/) It turned out Apple *did* have a point, but everyone could have handled the issue a lot better. In hindsight, the developer said, "I was lucky to have setup a direct way of distributing Dash a while ago and as a result I've been mostly unaffected by the removal from the App Store." (Source)

payments from a California-based company – but FastSpring handles payments to bank accounts in the European Union perfectly.

From a customer's perspective, paying for software works well, too. Their list of payment methods[7] is huge. As I said above, credit cards aren't common in Germany, so people will be happy to be able to resort to PayPal or wire transfer if you enable these options.

Ironic Software use Kagi and FastSpring next to each other: having two stores (plus the Mac App Store) surprisingly results in more sales for them. FastSpring is Ironic's Tom Andersen's favorite, though, because the backend is so nice to use.

Now Paddle is a different kind of competitor. Paddle's service centered around developer integration from the start. FastSpring is a general e-commerce service provider. FastSpring is aware of software developer's needs, but not everything they do revolves around folks like you and me.

Paddle offers a simple SDK for trials, in-app purchases, and license verification. Their service seemed to fit perfectly, so I gave it a spin in 2013. I tried the SDK in my first shipping app and learned a lot along the way – but I didn't feel comfortable because of the lack of *control*. I couldn't reliably find out how their stuff is working. There was no way to issue test purchases from within the app to see if the right events are raised. So in the end I didn't use Paddle at all.

FastSpring offers a test store front on the web and from within the app if you do in-app purchases. That's very convenient to see if your checkout works and if your app transitions from "locked" to "paid".

Although the amount of control FastSpring offers will make your job as a developer harder at first, once you nailed it and connect the dots it's dead simple to use.

That's what this book is for: **get all the obstacles out of your way so you can start using FastSpring in your app *today*.**

You don't have to figure out the edge cases. Let this book and the sample code take care of the details so you can focus on developing your product and selling it online. You can still tweak the setup later if you want.

[7] http://www.fastspring.com/payment-methods-and-security

Getting a Few Technical Terms Straight

We'll be talking about "keys" and "codes" a lot. To prevent confusion, I want to prime you with this list of important technical terms.

FastSpring
: That's your e-commerce solution. It's your storefront and product management system. It is also the hub of your own little customer support center where order details are stored. After all, you'll be the person in charge when people request refunds.

CocoaFob
: A set of algorithms, available as Open Source scripts, to generate and verify license codes. It's supported by FastSpring and very easy to implement.

  Source is available at: https://github.com/glebd/cocoafob

Public Key
: OpenSSL-generated random number used for 3rd parties to verify the data comes from you. Used for e-mail to tell recipients it was really you who sent the mail. Used for license code verification in our case.

Private Key
: OpenSSL-generated random number you have to, well, keep private. It's needed to create and sign the license key data. Should be kept private because you can infer the public key from it.[8] Still, we're going to need to give it to FastSpring for license code generation. So don't re-use this private key for anything else. We'll discuss this further in the next chapter.

License Code
: The string of characters your customers will receive after they buy your app.

  CocoaFob-generated license codes should look familiar if you ever bought a Mac app outside the App Store. They tend to look like this:

[8] Thanks to Jorge D. Ortiz Fuentes (http://powwau.com/en/) for corrections of the key descriptions (and a lot more). For encryption/decryption of information, both keys can be used for both actions. But always keep the private key secure.

```
GAWQE-FCUGU-7Z5JE-WEVRA-PSGEQ-Y25KX-9ZJQQ-GJTQC-
CUAJL-ATBR9-WV887-8KAJM-QK7DT-EZHXJ-CR99C-A
```

License Name
: Also called "licensee," the name of your customer your app is licensed to. We will set up license code generation to be based on the customer's name, but you can remove this information from the process. We'll talk about that later.

License Code Verification
: The process you have to implement in your app: the user enters a license code, the app uses the public key to decrypt the information from it, then it verifies if the information matches some internal criteria.

  Verified license codes result in unlocking the application, providing additional features, or ending a time-based trial.

Licensing Is All About a Big State Change

Copy protection of applications through licensing is all about handling transitions from unactivated to activated. Since there's going to be quite a bit of code involved in implementing this, it pays off to do it in a clean way and represent app state expressively.

The state the app is in is determined at launch, for example. But changes can happen over time when the trial period is up or the user enters a license code to unlock the app. The state representation changes have to be handled by your application: you have to handle transitions.

With Swift, it's trivial to represent the state as an enum with an associated value for convenience:

```
1  enum LicenseInformation {
2      case Registered(License)
3      case Unregistered
4  }
```

Or, if you want to support a time-based trial:

```
enum LicenseInformation {
    case Registered(License)
    case OnTrial(TrialPeriod)
    case TrialUp
}
```

That's the foundation of everything we're going to do: giving the application a means to find out which state it's in at the moment, and handle transitions.

The associated `License` type is based on license code and licensee name for the sample apps of this book:

```
struct License {
    let name: String
    let licenseCode: String
}
```

You're invited to stick to that and use it as-is in your applications. It's your choice if you want different details to be captured, though. You'll simply have to replace the type with something else and adjust the objects which read and write licenses accordingly.

Here's the battle plan:

We will start with setting up the app in your FastSpring dashboard in the next chapter. Then we'll implement a simple license check in the app's startup routine which determines if the app is allowed to run or if the user only sees the dialog to enter license details. In the last chapter, we're going to add a trial mode to enable users to test drive our software before they decide to buy.

Other Books by Me

Clean Cocoa is my mission: bringing you the best coding practices to create working applications for Mac and iOS.

To that end, I've written the following other books which you might want to check out:

- Exploring Mac App Development Strategies[9]
- Creating Multi-Process Mac Applications[10]

[9] https://leanpub.com/develop-mac-apps-clean-architecture-swift
[10] https://leanpub.com/create-multi-process-mac-apps

Set up Your App for Sale on FastSpring

Head to FastSpring's sign-up page[11] if you don't have an account already. Account activation may take a little while, so better do it early.

The FastSpring backend will take care of all the license code generation during the checkout process. This is part of the "order fulfillment" we're going to setup next.

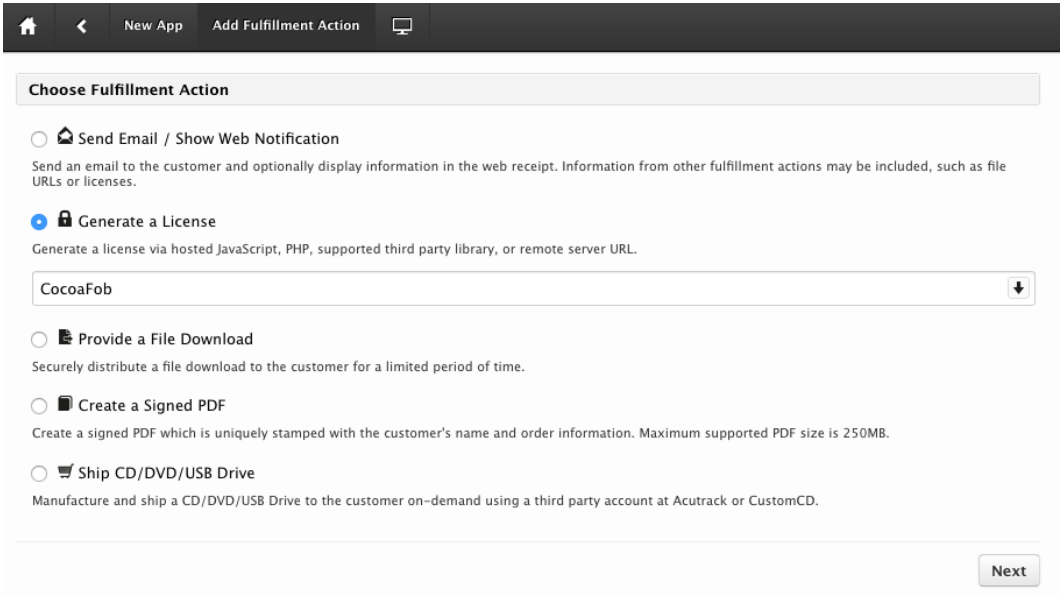
If you have an account set-up, great. Proceed!

If you still wait for your account approval, you may want to skim the following sections to get a feel for the process and come back to it later.

This step doesn't have to be completed before you can work in Xcode. Up next, we'll take a close look at implementing license verification in code. Start there while you wait.

[11] http://www.fastspring.com/sign-up

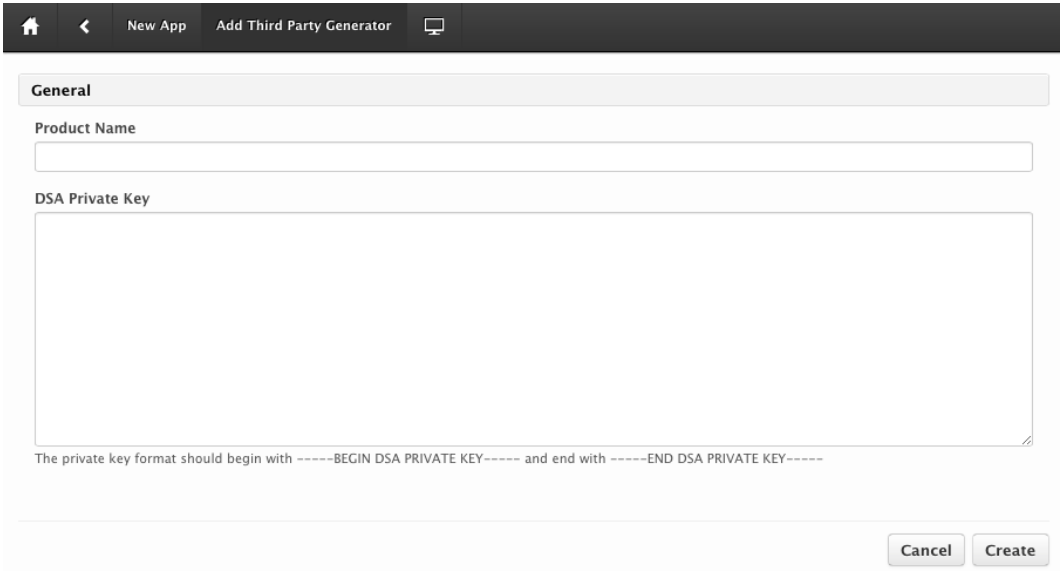
Configure License Code Generation via CocoaFob



Choose CocoaFob-based license generation from the fulfillment options

You will find license code generation under “Fulfillment Actions” at the bottom.

There’s a plethora of actions to choose from. You can send customizable e-mails with the license code (which I recommend you do, see the appendix), you can sign an e-book with the customer’s name as a means to protect the PDF from being copied, or you can ship a volume with the software as a physical object. I find PDF signing pretty cool if you want to sell e-books and worry about file-sharers. Then again there are websites which buy books, sign them with their URL, and give the copy away for free. They probably monetize through ads. So don’t expect too much protection from this.



The license generation setup form

Set up license generation:

- Add a fulfillment action,
- pick “Generate a License” from the options, and select CocoaFob.
- Hit “Next”.
- Pick a name for CocoaFob to use for the licenses.
- We will provide a value for the “DSA Private Key” thing in just a moment.

The name doesn’t have to be the same as your FastSpring product ID or product name (`newapp` and “My New App” in my case, respectively). It can be an arbitrary string to your liking. You can use the product name field of the license generator to include versioning information or whatever you want to differentiate generated licenses.

Potential hackers will probably have an easier time scanning your binary for “newapp” or a variant of your app’s name than, say, “/g}eJ3U8bG29r+wfcZ;” when they want to crack the license validation. We developers are humans and prefer readable names. But in the end the value of the name field doesn’t really matter.

Consider both options. Remember that every action you take also sends a message to your self. Which message do you want to send yourself: being paranoid about being hacked, securing access to your goods, or giving the gift of your app to the world? I personally want to spread my stuff; I'm not worried about people cracking my apps because those who really use it professionally will pay anyway; and I want to teach people that buying and supporting developers matters (in case they didn't know). That's why I don't create freeware applications most of the time. In the end, it's all about your values. Better be honest with yourself.

Book Sample Information

Only a selection of sections are added to this chapter in this book sample. I selected sections which I find most interesting to show you what is covered in the book. Because the sample is missing context, a few things might not make sense the way it is.

In the full book, you'll learn how to:

- use the embedded in-app store of FastSpring to buy license codes,
- offer additional in-app purchases,
- support app activation via e-mail,
- and much more!

Sample App 1: How to Verify License Codes in Your App

According to the license template `#{product},#{name}` the license code is just an encrypted form of, for example, "`MyNewApp,Christian Tietze`".

A license code is invalid when checked against that template if it satisfies at least one of the following criteria:

- It's not actually a string encrypted by a private key matching the public key known to the app.
- The extracted application name doesn't match or is missing.
- The extracted licensee name doesn't match the name the user provides or is missing.

We need to embed the CocoaFob algorithms and create an interface like the one pictured above to actually consume license information in the app. This way users will unlock the app's functionality after a purchase.

Tyler Hall researched the rates of piracy of his apps and found out 83% of the users of VirtualHostX were using a pirated copy. Piracy was strong for quite some time – until he added server-side validation to unlock license codes. I remember how badly gamers reacted when single-player computer games required an internet connection on first launch. That was a lot of years ago. Nowadays you can assume permanent connectivity for most of your customers.

In his blog post[12] on piracy, Tyler listed a few other measures against software piracy that did lower the number a bit, too, although not by much. Bugging the user with a personalized guilt-laden information dialog converted about 5%. Giving pirates a one-time discount to convert to real customers worked for about 11%. Remember what Craig Scott said about turning angry prospects into happy customers? It seems to work with people who want to use your app for longer than just the trial period, too. The deal has to be good from their point of view.

You can find a fully functional sample app in the book's code repository. It's in the `No-Trial-Verify-at-Start` folder.[13]

Embedding CocoaFob

For Swift 3, CocoaFob provides a native implementation. If you use Objective-C, refer to the instructions in the appendix.

Manual Checkout

I tend to add GitHub-based projects as submodules to simple projects like these. That's how the sample code[14] works, too. I put all of them into an "External" folder. So from the project root of your project, these are the commands you'd have to enter at the command line:

[12] http://tyler.io/experimenting-with-piracy/

[13] https://github.com/CleanCocoa/mac-licensing-fastspring-cocoafob/tree/master/No-Trial-Verify-at-Start

[14] https://github.com/CleanCocoa/mac-licensing-fastspring-cocoafob

```
$ mkdir External
$ cd External/
$ git submodule add git://github.com/glebd/cocoafob.git
```

Alternatively, you can download the code from GitHub[15] and add the files manually.

You're going to need the following files from the `swift/CocoaFob` subfolder:

- `CFUtil.swift`
- `CocoaFobError.swift`
- `CocoaFobLicVerifier.swift`
- `CocoaFobStringExt.swift`

Add them to your Xcode project. That's everything you need to do to prepare the project.

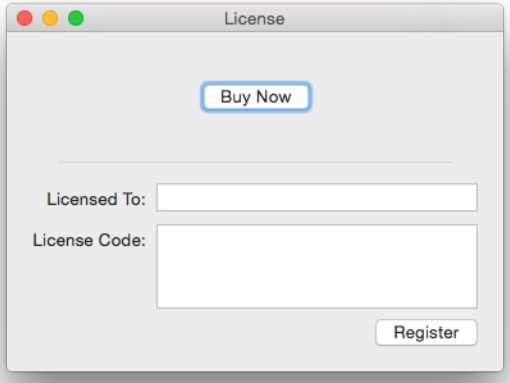
Dependency Managers

CocoaFob now also provides CocoaPods integration. The resulting library consists of all the stuff you are going to need – plus a bit more, like license key generation.

As of early 2017, Carthage and the Swift Package Manager are not supported.

[15] https://github.com/glebd/cocoafob/tree/master/swift/CocoaFob

Obtain License Information from the User



The "Enter a License" window of New App.

With the `LicenseVerifier` service object we are able to know if license information work. Your application will need to provide a form for the user to provide this information in the first place, though.

Most commonly, there'll be a "License Product …" main menu item where users can enter and review their license information. Some applications utilize a preference pane for that purpose. It's up to you where you place it. For the *NewApp* sample application, I chose a dedicated license window.

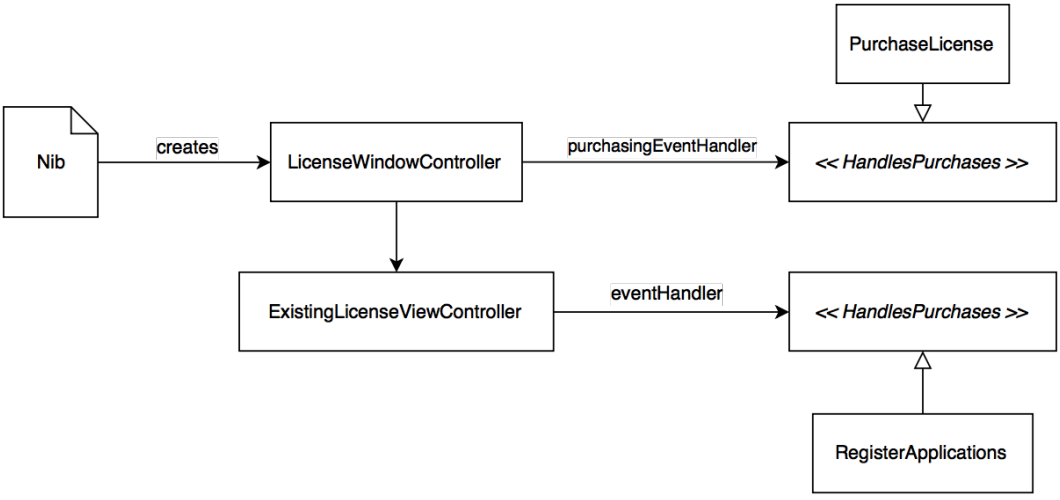
Creating a window like the one pictured above is very easy in Xcode. You can even use Cocoa Bindings to populate the form fields with values from `NSUserDefaults` where the sample app stores license information.

Be wary about using Cocoa Bindings, though: changes to the text fields' contents are stored immediately. The user can change values later and even without pressing the button invalidate the license info. You have to check that upon launch at least.

Even better, don't use Cocoa Bindings in a shipping application. Instead, use the traditional round-trip through view controllers. Store valid information only, and use an instance of `LicenseVerifier` to ensure that.

Window and Event Handler Architecture

So you can use the code for your real-world application, I didn't just throw together working components but constructed clean and pluggable components.



LicenseWindowController and related objects

`LicenseWindowController` is created through the Nib. It handles pressing the "Buy Now" button and delegates to a `purchaseEventHandler` implementing its `HandlesPurchases` protocol. We'll display a store in the `PurchaseLicense` service object which implements the protocol.

`ExistingLicenseViewController` handles form input and reacts to the "Register" action. It's owned by `LicenseWindowController`. It delegates license verification through its `HandlesRegistering` protocol to the `RegisterApplication` service.

We focus on `ExistingLicenseViewController` here to handle user input and look at `PurchaseLicense` later.

```
public protocol HandlesRegistering: class {
    func register(name: String, licenseCode: String)
}

public class ExistingLicenseViewController: NSViewController {
    @IBOutlet public weak var licenseeTextField: NSTextField!
    @IBOutlet public weak var licenseCodeTextField: NSTextField!

    public var eventHandler: HandlesRegistering?

    @IBAction public func register(sender: AnyObject) {
        guard let eventHandler = eventHandler else {
            return
        }

        let name = licenseeTextField.stringValue
        let licenseCode = licenseCodeTextField.stringValue

        eventHandler.register(name, licenseCode: licenseCode)

    }

    public func displayEmptyForm() {
        licenseeTextField.stringValue = ""
        licenseCodeTextField.stringValue = ""
    }

    public func displayLicense(license: License) {
        licenseeTextField.stringValue = license.name
        licenseCodeTextField.stringValue = license.key
    }
}
```

That's a very simple approach. This is what view controllers ought to do: react to user input and delegate to other objects as soon as possible.

This kind of approach to implementing view controllers makes them easily testable. In fact, the sample code contains more than 50 tests with quite a few regarding Nib

loading and view controller interaction.

Register the Application and Representing License Information

It is apparent that `RegisterApplication`, which implements the `HandlesRegistering` protocol, is way more interesting.

```
public class RegisterApplication: HandlesRegistering {
    let licenseVerifier: LicenseVerifier
    let licenseWriter: LicenseWriter
    let changeBroadcaster: LicenseChangeBroadcaster

    public convenience init() {
        self.init(licenseVerifier: LicenseVerifier(),
            licenseWriter: LicenseWriter(),
            changeBroadcaster: LicenseChangeBroadcaster())
    }

    public init(licenseVerifier: LicenseVerifier,
            licenseWriter: LicenseWriter,
            changeBroadcaster: LicenseChangeBroadcaster) {
        self.licenseVerifier = licenseVerifier
        self.licenseWriter = licenseWriter
        self.changeBroadcaster = changeBroadcaster
    }

    public func register(name: String, licenseCode: String) {
        if !licenseVerifier.isValid(licenseCode: licenseCode,
                                    forName: name) {
            displayLicenseCodeError()
            return
        }

        let licenseInformation = LicenseInformation.registered(
            License(name: name, key: licenseCode))

```

```
30          licenseWriter.store(licenseCode: licenseCode, forName: name)
31          changeBroadcaster.broadcast(licenseInformation)
32      }
33
34      func displayLicenseCodeError() {
35          Alerts.invalidLicenseCodeAlert()?.runModal()
36      }
37  }
```

You know `LicenseVerifier` already. `LicenseWriter` is a simple class which stores the license information into user defaults while another object of type `LicenseProvider` would be responsible for reading these values out again.

Attempting to register and unlock the app fails if `LicenseVerifier` doesn't succeed for any of the reasons we got to know already, most commonly because name and license code don't work out.

The `RegisterApplication` service object then creates the proper license information and stores it so the app stays unlocked on subsequent starts.

The `LicenseChangeBroadcaster` wraps constructing a notification from the `LicenseInformation` enum and its associated `License` value object and sending it to a `NSNotificationCenter`.

The value objects we use behind the scenes are, simplified:

```
1  struct License {
2      let name: String
3      let licenseCode: String
4  }
5
6  enum LicenseInformation {
7      case unregistered
8      case registered(License)
9  }
```

That's all there is to representing if a user has unlocked the app already.

The notification `LicenseChangeBroadcaster` fires can be consumed by any interested service object in your app to change display values or unlock features.

In the sample application, which does not allow using the main application interface until the app is unlocked, `AppDelegate` is the gatekeeper. It subscribes to these events and unlocks the main app:

```
func licenseDidChange(_ notification: Notification) {

    guard let userInfo = notification.userInfo,
        let licenseInformation = LicenseInformation.fromUserInfo(userInfo)
        else { return }

    switch licenseInformation {
    case .registered(_):
        displayThankYouAlert()
        unlockApp()

    case .unregistered:
        // If you support un-registering, handle it here
        return
    }
}
```

With this and the view controller–event handler architecture from above, we're able to display the license info form, register the app, and unlock features without tight coupling.

In fact, you can put this code inside your own application, change a few parameters, and simply have to react to the license change event somewhere. That's all there is in terms of the most basic copy protection.

Book Sample Information

Only a selection of sections are added to this chapter in this book sample. I selected sections which I find most interesting to show you what is covered in the book. Because the sample is missing context, a few things might not make sense the way it is.

In the full book, you'll learn how to:

- use the embedded in-app store of FastSpring to buy license codes,
- offer additional in-app purchases,
- support app activation via e-mail,
- and much more!

Sample App 2: Adding a Time-Based Trial

Licensing now just works. But having to buy an app without test-driving it is not very comfortable for your users. Especially if your application costs more than $10, which may be considered to be the threshold of impulse purchases, users will want to find out if your product really is for them.

That's where trials come into play. You offer new users a test drive for a limited time so they can decide if they want to keep using the product. Your trial period should be long enough to make your app part of your user's lives or they will shrug the purchase dialog off too easily.

To add a time-based trial to your app, you have to change `LicenseInformation` a bit and add a third intermediate state:

```
1  enum LicenseInformation {
2      case registered(License)
3      case onTrial(TrialPeriod)
4      case trialUp
5  }
```

I experimented with different `TrialPeriod` implementations.

To use a start date and a duration value came natural, but it became unnecessarily clumsy to use. The duration has to be provided by the app in code anyway. And for most timer operations, you have to compute the end date of the trial period, too. So I suggest this instead, in the most basic form:

```
struct TrialPeriod {
    public let startDate: Date
    public let endDate: Date
}
```

From there on, this chapter will teach you how to change the existing application to take the trial into account.

You can find a fully functional sample app incorporating a trial mode in the book's code repository. It's in the `Trial-Expire-While-Running` folder.[16]

Trial Information I/O

You have to store the `TrialPeriod` somewhere. The sample app will put trial information into the `NSUserDefaults`.

The user defaults is a pretty bad place for a shipping app if you worry about users trying to avoid to pay for your software since savvy users can manipulate the values via command line.

```
$ defaults read de.christiantietze.MyNewApp
{
    "trial_ending" = "2015-08-30 09:17:25 +0000";
    "trial_starting" = "2015-08-25 09:17:25 +0000";
}
```

Better put it someplace safer if you're concerned about this. Another option that's probably overkill for most of us: require users to register on the web and store the trial duration on your server.

Keep in mind though that it likely won't pay off to make it harder for cheap users to extend the trial or hack the license mechanism. Every hour you spend on copy protection against a handful of people is an hour you won't spend improving the app for paying customers.

We used `NSUserDefaults` for storage of license information already. I wrote `TrialProvider` and `TrialWriter` in a similar way, mostly replacing the defaults keys.

[16] https://github.com/CleanCocoa/mac-licensing-fastspring-cocoafob/tree/master/Trial-Expire-While-Running

```
public class TrialProvider {
    public init() { }

    lazy var userDefaults: NSUserDefaults =
        UserDefaults.standardUserDefaults()

    public var currentTrialPeriod: TrialPeriod? {
        if let startDate = userDefaults.object(forKey:
                "\(TrialPeriod.UserDefaultsKeys.startDate)") as? Date,
            let endDate = userDefaults.object(forKey:
                "\(TrialPeriod.UserDefaultsKeys.endDate)") as? Date {

            return TrialPeriod(startDate: startDate, endDate: endDate)
        }

        return .None
    }
}

public class TrialWriter {
    public init() { }

    lazy var userDefaults: NSUserDefaults =
        UserDefaults.standardUserDefaults()

    public func storeTrial(trialPeriod: TrialPeriod) {
        userDefaults.set(trialPeriod.startDate,
            forKey: TrialPeriod.UserDefaultsKeys.startDate.rawValue)
        userDefaults.set(trialPeriod.endDate,
            forKey: TrialPeriod.UserDefaultsKeys.endDate.rawValue)
    }
}

extension TrialPeriod {
    public enum UserDefaultsKeys: String, CustomStringConvertible {
        case startDate = "trial_starting"
        case endDate = "trial_ending"

```

```
39          public var description: String { return rawValue }
40      }
41  }
```

Using the Clock to Obtain Time

The `TrialPeriod` value object encapsulates both start and end time. And we're able to read and write it to disk via `UserDefaults`.

A trial shall have a duration of 5 days for this app. To create a `TrialPeriod`, we need the current time as `startDate` and the duration to calculate `endDate`.

The current time can be obtained using `Date()`, the default constructor. But that's not helping us during tests, which will make the behavior of a whole bunch of service objects hard to verify – or worse, since nothing can be tested, everything ends up in one big convoluted mess.

Thinking in terms of an expressive domain and testing seams, we need a clock.

```
1  class Clock {
2      func now() -> Date {
3          return Date()
4      }
5  }
```

To make testing a bit more comfortable, I favor a protocol to supply test doubles instead of subclassing `Clock`:

```
public protocol KnowsTimeAndDate: class {
    func now() -> Date
}

public class Clock: KnowsTimeAndDate {
    public init() { }

    public func now() -> Date {
        return Date()
    }
}

public class StaticClock: KnowsTimeAndDate {
    let date: Date

    public init(clockDate: Date) {
        date = clockDate
    }

    public func now() -> Date {
        return date
    }
}
```

This helps to test `TrialPeriod` creation:

```
extension TrialPeriod {
    public init(numberOfDays daysLeft: Days, clock: KnowsTimeAndDate) {
        startDate = clock.now()
        endDate = startDate.addingTimeInterval(daysLeft.timeInterval)
    }
}
```

It's trivial to verify that `TrialPeriod(numberOfDays:, clock:)` actually adds the appropriate time interval to the current time in tests:

```
class TrialPeriodTests: XCTestCase {

    let clockDouble = TestClock()
    let irrelevantDate = NSDate(timeIntervalSinceNow: 987654321)

    func testCreation_WithClock_AddsDaysToCurrentTime() {

        let date = NSDate(timeIntervalSinceReferenceDate: 9999)
        clockDouble.testDate = date
        // 10 days
        let expectedDate = date.dateByAddingTimeInterval(10 * 24 * 60 * 60)

        let trialPeriod = TrialPeriod(numberOfDays: Days(10), clock: clockDo\
uble)

        XCTAssertEqual(trialPeriod.startDate, date)
        XCTAssertEqual(trialPeriod.endDate, expectedDate)
    }

    // ...

    class TestClock: KnowsTimeAndDate {
        var testDate: Date!
        func now() -> Date {
            return testDate
        }
    }
}
```

Backed-up by `KnowsTimeAndDate`, it's easy to adds a few convenience methods on `TrialPeriod`, too, to encapsulate logic related to `TrialPeriods` right where it belongs:

```
extension TrialPeriod {
    public func ended(clock: KnowsTimeAndDate) -> Bool {
        let now = clock.now()
        return endDate < now
    }

    public func daysLeft(clock: KnowsTimeAndDate) -> Days {
        let now = clock.now()
        let timeUntil = now.timeIntervalSinceDate(endDate)
        let daysUntil: Double = fabs(Days.amountFromTimeInterval(timeUntil))

        return Days(daysUntil)
    }
}
```

These two methods are a pain to test if you substitute `clock.now()` with `Date()`: when you run your tests, the date values can be off a second and cause trouble in assertions.

`Days` is just a simple wrapper around double values which helps to calculate seconds (`TimeInterval`) without actually passing meaningless integers around. With Swift's type checking, it then becomes easy to work with durations and enforce validity of the parameters.

```
public struct Days {
    public static func timeInterval(amount: Double) -> Double {
        return amount * 60 * 60 * 24
    }

    public static func amount(timeInterval: TimeInterval) -> Double {
        return timeInterval / 60 / 60 / 24
    }

    public let amount: Double

    /// Rounded to the next integer.
    public var userFacingAmount: Int {
```

```
14
15          return Int(ceil(amount))
16      }
17
18      public init(timeInterval: TimeInterval) {
19          amount = fabs(Days.amount(timeInterval: timeInterval))
20      }
21
22      public init(_ anAmount: Double) {
23          amount = anAmount
24      }
25
26      public var timeInterval: TimeInterval {
27          return Days.timeInterval(amount: amount)
28      }
29
30      public var isPast: Bool {
31          return amount < 0
32      }
33  }
```

In my book on application architecture, *Exploring Mac App Development Strategies*, I talk about the benefit of value objects and how to make things testable in more detail. You can find more about that book and value objects on my website.[17]

The main lesson I want to uncover is this: try to defer object creation to service objects like the good old Factory object. Especially creation of things you do not own, like `Date` and similar Foundation types. It pays off to not worry about their behavior when you write tests (which you should).

Now that `Days` takes care of encapsulating the main time interval we'll need, and `Clock` to tell the current time, `TrialPeriod` becomes super simple:

[17] http://cleancocoa.com/

```
extension TrialPeriod {

    public func ended(clock: KnowsTimeAndDate) -> Bool {
        let now = clock.now()
        return endDate < now
    }

    public func daysLeft(clock: KnowsTimeAndDate) -> Days {
        let now = clock.now()
        let timeUntil = now.timeIntervalSince(endDate)

        return Days(timeInterval: timeUntil)
    }
}
```

Book Sample Information

Only a selection of sections are added to this chapter in this book sample. I selected sections which I find most interesting to show you what is covered in the book. Because the sample is missing context, a few things might not make sense the way it is.

In the full book, you'll learn how to:

- use the embedded in-app store of FastSpring to buy license codes,
- offer additional in-app purchases,
- support app activation via e-mail,
- and much more!

Appendix

There are some things which don't naturally fit into the other chapters of this little book, but which are interesting to make your app launch outside the Mac App Store a success.

Next to some resources on the topic, there're chapters on bundle sales and in-app purchases you don't want to miss.

Selling outside the Mac App Store has its flip-side: you don't have that central global warehouse at your disposal. But on the other hand you have the opportunity to team up with other indie devs and sell cool app bundles. You may even be as crazy as bundling apps, e-books, and other media!

Good luck trying that on the App Store.

We'll dig into bundles, in-app purchases, and other technical details in the following sections.

Book Sample Information

Only a selection of sections are added to this chapter in this book sample. I selected sections which I find most interesting to show you what is covered in the book. Because the sample is missing context, a few things might not make sense the way it is.

In the full book, you'll learn how to:

- use the embedded in-app store of FastSpring to buy license codes,
- offer additional in-app purchases,
- support app activation via e-mail,
- and much more!

Other Books by Me

Clean Cocoa is my mission: bringing you the best coding practices to create working applications for Mac and iOS.

To that end, I've written the following other books which you might want to check out:

- Exploring Mac App Development Strategies[18]
- Creating Multi-Process Mac Applications[19]

[18] https://leanpub.com/develop-mac-apps-clean-architecture-swift
[19] https://leanpub.com/create-multi-process-mac-apps