iOS in the Enterprise

A hands-on guide to managing iPhones and iPads

John Welch

iOS in the Enterprise

A hands-on guide to managing iPhones and iPads

John Welch

Peachpit Press

iOS in the Enterprise: A hands-on guide to managing iPhones and iPads
John Welch

Peachpit Press

1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at: www.peachpit.com
To report errors, please send a note to errata@peachpit.com
Peachpit Press is a division of Pearson Education

Editor: Nancy Peterson
Production editor: Myrna Vladic
Development editor: Bob Lindstrom
Copyeditor: Darren Meiss
Cover design: Aren Straiger
Interior design: Mimi Heft
Compositor: David Van Ness
Indexer: Joy Dean Lee

ISBN 13: 978-0-132-73603-9
ISBN 10: 0-132-73603-9

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

This book, like everything I do, is dedicated to the family I live with:
my amazing, beautiful, talented wife Melissa, and my son Alex,
who is about to go into the world as a grownup.

It's also dedicated to the family I don't live with who keep me sane:
Mom, Dad, Gypsye, Nicci, Mo, Brad, Kelly, Mark, Virginia, Jenny,
Michelle, Rachel, Ernie, Sami, Sly . . . you guys are all amazing,
and I'm lucky to know one of you, much less all of you.

ACKNOWLEDGEMENTS

The very concept that I did this even slightly alone is ridiculous. There are quite a few people without whom this book would not have happened, and I would be far, far crazier than I am:

To the best editing team ever, Nancy Peterson and Bob Lindstrom, who kept me focused, working and regularly laughing. (Seriously, Bob has some of the funniest editorial comments ever and they make a rather tedious task a lot more fun.) Nancy had the unenviable job of chief whip-cracker to someone who is really good at procrastination and she did it perfectly. Whatever shreds of a schedule we managed to keep were all due to her fantastical fanatical work. I am also deeply appreciative that they, (and Peachpit) not only allowed, but encouraged me to keep my "voice" throughout the book.

The Apple iOS team, without whom I'd have nothing to write about.

Sal and the AppleScript team, because any chance I have to thank one of the best groups at Apple, or anywhere, I will.

The folks at the/zimmerman/agency, in particular my boss, Mike, along with Curtis & Carrie: you've created the environment that let me experiment and learn how to do things with iOS that gave me the ability to write this book based on the real world experiences I've gained with Z. Thank you all for that and for not letting the agency become just another place to work. Everyone at Z, you guys are the best.

Zach, Chip, Lance, and all the folks at JAMF software who answered questions and provided extensions to demo keys and were absolutely invaluable as a resource, you guys have earned every dime you've made or ever shall make.

Jessica, the most awesome, wonderful, amazing former editor ever, who gave me my start in getting paid to write ... see what you started? Oh, and I have a lovely yard full of love bugs should you ever visit :-P

Kathy Moran, Paul Kent, Ron Moreau, Arek, Kevin, Ben, and all the other folks who work their keisters off to put Macworld Expo and MacIT together—thanks for letting me play too; you're all wonderful.

My brothers in arms, Peter and Darby … guys, WHAT is going on, and how much fun is this? Every Tuesday for over two years, I get some of my sanity back.

Jason, Phil, Chris, the Dans, and all the folks at Macworld: I know how much of a pain my name on the site can be for you. But thank you for putting it there anyway. It's still awesome every time I see it.

Dave Hamilton, ChuckL, JeffG, Dori, Tom, and all the other Expo peeps … every year I get a big funky reunion with my favorite people. Y'all are why I still get excited about expo.

The Group which must not be named shall nonetheless be thanked. Thank you to all the people on the Internet and elsewhere who have gone through the pain of learning how to manage iOS stuff and took the time to share their experiences. It's folks like you that make the Internet worthwhile, far more than any NMD collective ever will.

Finally, to the baddest, funniest, coolest group of ladies I know: The Tallahassee RollerGirls. Derby. Rocks.

This book took, one way or another, my entire life to write and this is a TINY fraction of those who helped.

CONTENTS

PART III MOBILE DEVICE MANAGEMENT

INTRODUCTION

Those of you who have to deal with more than a handful of iPhones, iPads, or iPod Touches already know why you manage iOS devices. For everyone else, "manage" is not a short way to say "impose draconian control." Managing devices on your network, including iOS devices, not only makes your life easier, but should also make life easier for your users.

That's my core philosophy with regard to device management. In the end, device management *has* to make life easier for the user.

A happy side benefit to this is that when done right, it makes *your* life easier, too. When a user can personally take an iPhone from activation to full network integration in two to three steps and about five minutes, it frees you and that user to actually *do stuff with the gear*.

WHY MANAGE iOS DEVICES?

I think we should all be clear on what is meant by that phrase because this book is pretty much built around it. While "managing iOS devices" can suggest all sorts of draconian imagery, the reality is a bit more mundane.

When you run a business or an IT department, you have to care about your company's "stuff." If you have a small number of people, it's pretty easy to adopt a "live and let live" policy, so your management tasks may start and end with "Here's the address for the email server we use. Have a nice day."

But as your company grows, or if you have data that you need to control securely, then you need ensure that your data is set up and managed in a consistent, sane manner. Consider a small doctor's office. Even with just two or three employees, that office has to take data security *very* seriously or many, many regulatory and legal entities may come down on it like a ton of bricks.

So that's what management is about. You're ensuring that your iOS devices are set up in a way that is consistent and sane for your needs, whatever those needs may be. Some of you may never need to care about disabling cameras, for example, while others may need to lock down those snapshot lenses as tightly as possible. That's what this book is about: Helping you meet your iOS device needs whatever they may be.

WHO NEEDS THIS BOOK?

The short answer is "anyone who wants to better manage their iOS devices."

(By the way, throughout the book, I'll use "iOS devices" to refer to the entire family of Apple products that run on iOS. If I'm talking about a specific product, such as an iPad, then I'll do so. Trust me, referring to "iOS devices" beats the pants off of "iPhone, iPad, and/or iPod Touch." It's also gobs easier to type.)

The longer answer is about the same as the short answer with more details. No one profile perfectly covers everyone using iOS devices. *Everyone* is learning how to deal with Apple's portable devices, from five- or ten-person SOHO shops to Big Enterprise. This book is simply a collection of information to help you out, regardless of your level of iOS usage.

WHAT THIS BOOK IS

This book is, as true as I can make it, a reference source. It is designed to be of use to people across their ranges of need—from someone who just wants a guide to use iTunes and a USB cable to someone who needs to set up SCEP and MDM and talk to their back-end directory servers.

As much as is practically possible, this book tries to help all of you. I hope it does so in a way that will be of use past the current version of the iOS (which is v.4.3.x at the time of this writing). That means I'm going to cover a lot of principles; the general application of said principles; and use specific, focused examples to illustrate an application when it makes sense, or when I've found an app that's particularly neat or cool. (Yes, neat/cool counts in IT. You'd be amazed.)

WHAT THIS BOOK IS NOT

If you're looking for a cookbook of how-tos, I will tell you now, this is not the book for you. While such books have their place, I think that place is the Internet, where information updates can be done more quickly. I'm not just being smarmy here. The words you're reading were written six or more months ago. As a result, any how-to or step-by-step example included here will be similarly old. (What, you think editing my verbosity happens in a fortnight?) Do you really want to use a step-by-step setup that may be older than the iOS version you're trying to use it on? No.

In a sense, overly detailed step-by-step how-to books are handing you a fish. Instead, I want to teach you how to fish. This book is here to help you learn about what's going on with iOS devices and how they work with regard to iOS management, so that you can develop the exact way *you* wish to implement that management in *your* environment in a way that works for *you*.

THANKS

Outside of the specific thank-yous that are in the various prefaces to this book, I want to give some thanks specifically to: Apple, for the iOS, the devices, and the management APIs; Cisco, for SCEP; Microsoft, for giving Windows Server 2008 the ability to act as a SCEP server even though I doubt that iOS was the reason; JAMF, for giving people yet another reason to buy Casper (it really is an amazing product); and a host of people on the Internet who have contributed knowledge and help on this subject, in general and directly to me, because they felt that adding to the knowledge base is The Right Thing To Do. When I can nail the information down to one source, I'll make sure you get credit. This book is as much yours as mine.

WELCOME TO iOS IN THE ENTERPRISE

iOS is, of course, the operating system for Apple's iPad, iPhone, and iPod Touch. If you haven't heard of those devices, well, I'm not sure how you would not have heard of those and still be interested in this book. Anyway, iOS and the devices that run it are really awesome and cool; but when you have to manage all of them, some

THE TOOLS

You'll need to be familiar with a small set of tools and concepts to get the most out of this book and managing your iOS devices.

iTUNES

iTunes is one of Apple's two primary tools for managing iOS devices. In the consumer space, it is *the* primary tool, and every iOS device running iOS 4.x has to connect to iTunes via USB at least once. iTunes is a free download from Apple and runs on Windows or Mac OS X.

iPHONE CONFIGURATION UTILITY

The iPhone Configuration Utility (iPCU) is the other primary Apple-provided tool for managing iOS devices. It is designed for administrators who need to manage their devices beyond the capabilities of iTunes and the on-device options. The iPCU is a free download from Apple and runs on Mac OS X or Windows.

APPLESCRIPT

The book talks about using AppleScript to automate tasks involving the iPCU and various XML-based configuration files. AppleScript is Apple's own scripting language that uses vaguely quasi-English syntax. It is included with Mac OS X.

of that awesomeness may decrease. Fear not! This book is here to re-awesome-ize those devices, and help make you seem awesome as well. To help you in your awesome journey to Ultimate iOS Awesomeness, here are a few tidbits you'll want to know about upfront.

XCODE
Even if you aren't an iOS developer, if you plan to distribute in-house or "enterprise" apps, Xcode will be a necessary part of the process. Xcode is Apple's primary development environment and is included free on every new Mac and is also available from the Mac App Store for around $5 U.S.

A WEB SERVER
When we start talking about managing iOS devices on a large scale, or wirelessly, you'll need a web server. The platform and brand really don't matter. In fact, you don't even have to own the web server yourself. But, you will need one.

THE CONCEPTS

Along with the tools, some concepts and protocols have a lot to do with managing iOS Devices. A good foundation in these will make your life much easier.

SECURE SOCKETS LAYER (SSL)
SSL is heavily used when managing iOS devices, along with related concepts such as Public Key Infrastructure (PKI). It is a *really* good idea to have at least a conceptual grasp of basic SSL concepts, especially when dealing with SCEP and Mobile Device Management.

LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL (LDAP)
LDAP, while not a direct part of managing iOS devices, is the basis of pretty much any directory service you'll see or use, including OpenLDAP, Open Directory, and Active Directory. Many products for managing iOS devices have LDAP integration options, and a good understanding of LDAP and Directory Services in general will be a great help to you.

This page intentionally left blank

PART I

iTUNES AND iPHONE **CONFIGURATION** UTILITY

1

WHEN **iTUNES** IS **ENOUGH**

Contrary to what a lot of people may want you to think, you don't always need a specialized tool to manage iOS devices. When you have simple needs, all you require is iTunes. Sometimes, simple is good.

LIMITATIONS OF iTUNES

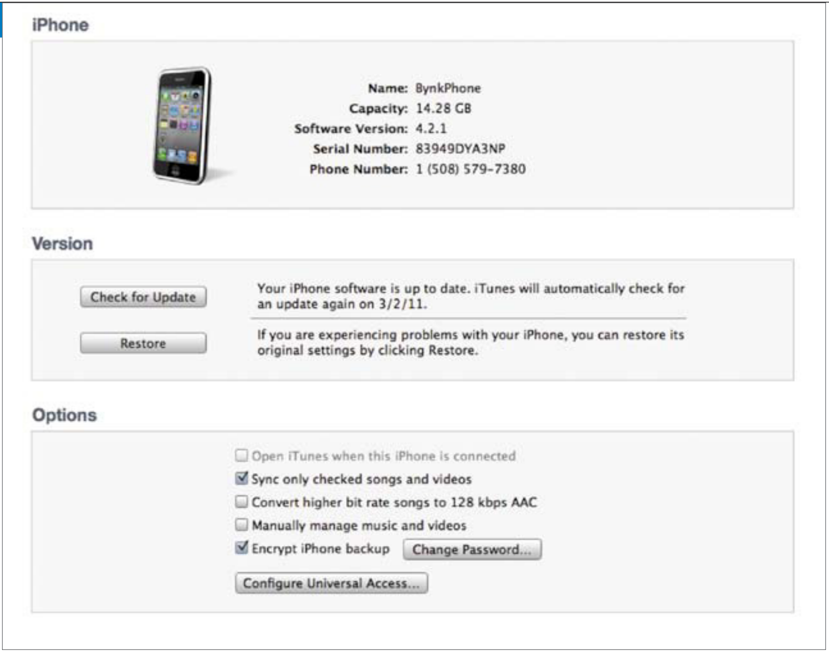
Of course, the downside of simple is that it's *simple*. Managing iOS devices with iTunes means that you're accepting a set of limitations over what you can manage and how you do so.

First, you have to use iTunes via USB. There's no option for over-the-air (OTA) configuration. Second, most of your control will come from the device itself, so the management process is fairly manual.

Realistically, an iTunes-only configuration is for the small office/home office (SOHO), or for the "small" end of small-to-medium business (SMB) markets. Still, it's great for small numbers of devices, or when people are using their personal devices for company purposes. If you have to configure a lot of devices, or you need more control, iTunes won't work so well.

NOTE: A security risk is always involved when using personal devices for company data. People leave companies and may not remember to wipe company data from their devices. Because every company is different with different needs, this is not a question I can answer for you in some generic way or with a clever bon mot. You'll want to seriously consider the kinds of data that users will store before you permit the use of personal devices.

MANAGING WITH iTUNES


iPhone

Name: BynkPhone
Capacity: 14.28 GB
Software Version: 4.2.1
Serial Number: 83949DYA3NP
Phone Number: 1 (508) 579-7380

Version

Check for Update — Your iPhone software is up to date. iTunes will automatically check for an update again on 3/2/11.

Restore — If you are experiencing problems with your iPhone, you can restore its original settings by clicking Restore.

Options

- [ ] Open iTunes when this iPhone is connected
- [x] Sync only checked songs and videos
- [ ] Convert higher bit rate songs to 128 kbps AAC
- [ ] Manually manage music and videos
- [x] Encrypt iPhone backup

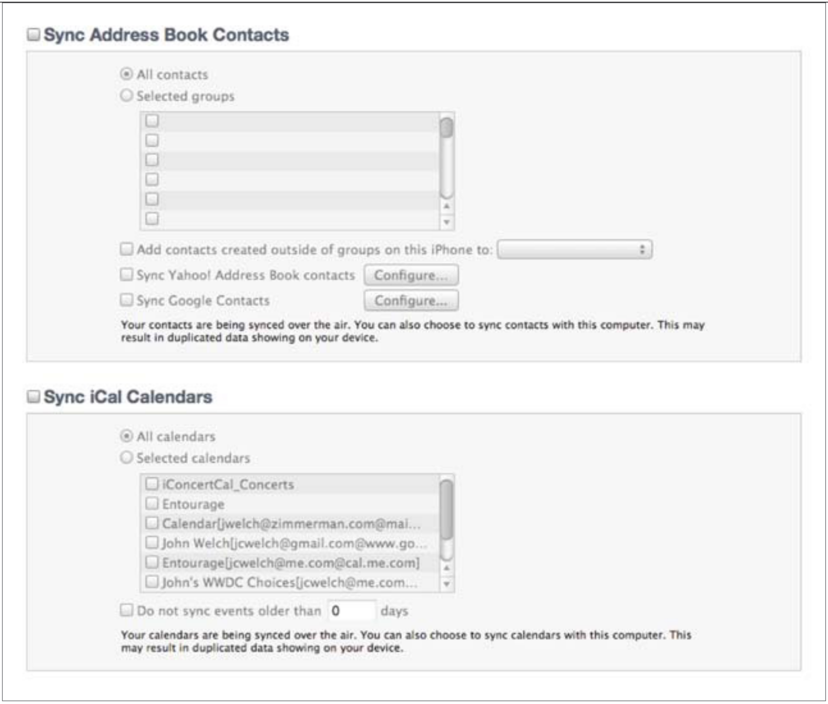
FIGURE 1.1 Device summary settings

So let's look at what you can get out of iTunes. In a nutshell, there's not a whole lot. The iTunes settings for iOS devices don't really revolve around limiting access, but rather managing how you use the devices. For example, in the device summary settings in **Figure 1.1**, you can see that the management options are pretty basic.

I recommend that you encrypt the backups for devices used with business data. iTunes offers handy, but not exactly high-end, management, and you have to set this up on the computer, not the device. (Oddly, this is where the general tediousness of using iOS devices with multiple computers works in your favor by discouraging users from modifying your setup. Trying to match settings between a home Mac and a work Mac—or even more bizarre, iTunes on Windows and iTunes on a Mac—is enough work that most people just won't bother.)

iTunes' application "management" settings are even *more* basic, to the point of not really being what many would think of as "management." They're not really intended to restrict your access to applications, or even control whether you can or cannot add applications to the device. Instead, they're really just there to help you set up how apps are laid out on the device, whether a specific app should be synced, and whether new apps should automatically be synced. That's it.

FIGURE 1.2 Contact and calendar settings



☐ **Sync Address Book Contacts**

◉ All contacts
○ Selected groups

☐ Add contacts created outside of groups on this iPhone to:
☐ Sync Yahoo! Address Book contacts
☐ Sync Google Contacts

Your contacts are being synced over the air. You can also choose to sync contacts with this computer. This may result in duplicated data showing on your device.

☐ **Sync iCal Calendars**

◉ All calendars
○ Selected calendars

☐ iConcertCal_Concerts
☐ Entourage
☐ Calendar[jwelch@zimmerman.com@mai...
☐ John Welch[jcwelch@gmail.com@www.go...
☐ Entourage[jcwelch@me.com@cal.me.com]
☐ John's WWDC Choices[jcwelch@me.com...

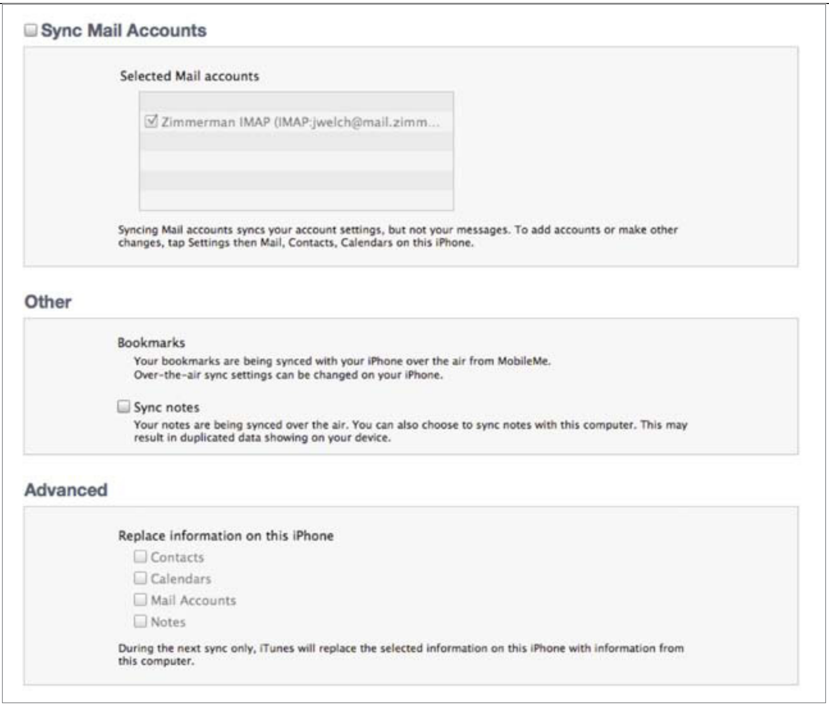
☐ Do not sync events older than 0 days

Your calendars are being synced over the air. You can also choose to sync calendars with this computer. This may result in duplicated data showing on your device.

That's not to say that iTunes' settings are useless for device management. For example, if you've ever tried to manually set up email accounts on an iOS device, you know that it's not the most pleasant experience. The iOS is rather insistent about not letting you skip any verification steps, no matter that you just want to enter the info and move on. iTunes provides an easy way to avoid a lot of this pain.

On the computer that will sync with the iOS device, set up all your email accounts, calendars accounts, and contacts in the iTunes Info sections (**Figure 1.2** and **Figure 1.3**).

Then, sync the device. Voila! All your account setup is done. Once that's done, you'll want to kill email sync within iTunes because the device will now handle that sync for you. However, you'll still need to manually sync calendars and contacts.



Syncing Mail accounts syncs your account settings, but not your messages. To add accounts or make other changes, tap Settings then Mail, Contacts, Calendars on this iPhone.

Your bookmarks are being synced with your iPhone over the air from MobileMe. Over-the-air sync settings can be changed on your iPhone.

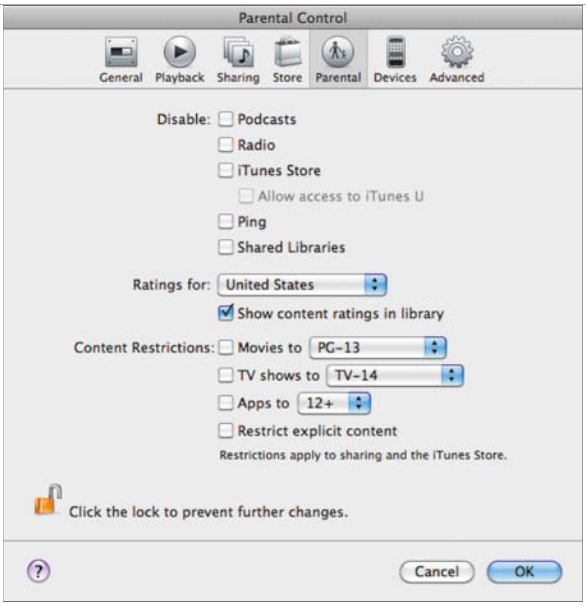
Your notes are being synced over the air. You can also choose to sync notes with this computer. This may result in duplicated data showing on your device.

During the next sync only, iTunes will replace the selected information on this iPhone with information from this computer.

FIGURE 1.3 Mail and other settings

One more point about email. If you or your employees are going to check email from computers and iOS devices, you really, really want to use IMAP standards for email, and not POP. IMAP is designed for this kind of use, POP is not. Yes, POP has that "leave it all on the server" setting; however, just like putting a big spoiler on your Civic does not turn it into a Porsche 917, leaving POP email on the server does not turn it into IMAP.

NOTE: If you're using CalDAV for calendaring or CardDAV/LDAP for contacts, you won't need to sync manually. However, you won't be able to use iTunes to sync those apps, not even to set them up. For whatever reason, Apple does not to allow you to sync CardDAV accounts. You can sync a CalDAV account via iTunes, but it won't create a CalDAV account on your Mac the way Mail creates accounts.


Parental Control

Disable: [ ] Podcasts
[ ] Radio
[ ] iTunes Store
[ ] Allow access to iTunes U
[ ] Ping
[ ] Shared Libraries

Ratings for: United States
[✓] Show content ratings in library

Content Restrictions: [ ] Movies to PG-13
[ ] TV shows to TV-14
[ ] Apps to 12+
[ ] Restrict explicit content
Restrictions apply to sharing and the iTunes Store.

Click the lock to prevent further changes.

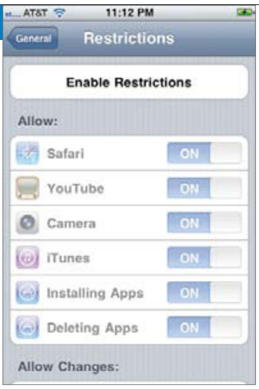
FIGURE 1.4 iTunes parental controls

The IMAP standard includes a lot of features that work well on devices such as the iPhone and the iPad, and POP does not. Accessing the same email account from multiple places is what IMAP was designed for, and using it will make your life much easier.

I'm not saying that iTunes is *completely* useless for restricting/controlling what can be done with iOS devices. It's not, but we need to keep in mind that iTunes' definition of "management" is simply different from ours. In the iTunes Parental controls (**Figure 1.4**), for example, you can do a few things to keep people out of mischief.

You can disable access to podcasts, the iTunes Store, Ping, and you can set content restrictions. However, these limits are for iTunes, not iOS devices. It just happens that when you use iTunes to sync/manage the devices, this has some happy side effects. For example, if you can't install apps or podcasts in iTunes, it's a bit hard to install them on say, an iPad. But that's not really an awesome way to do things. Luckily for us, we have an alternative method to use here: the device settings.

USING **DEVICE SETTINGS**

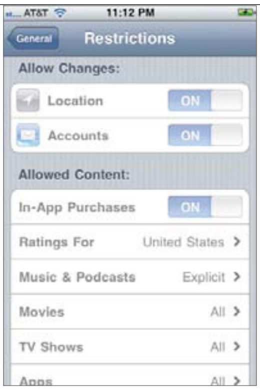

AT&T 11:12 PM
General Restrictions

Enable Restrictions

Allow:

Safari ON
YouTube ON
Camera ON
iTunes ON
Installing Apps ON
Deleting Apps ON

Allow Changes:

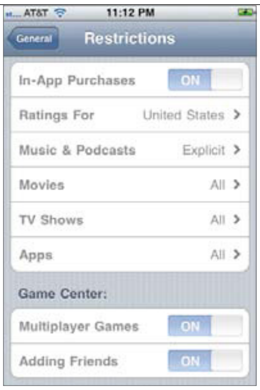

AT&T 11:12 PM
General Restrictions

Allow Changes:

Location ON
Accounts ON

Allowed Content:

In-App Purchases ON
Ratings For United States >
Music & Podcasts Explicit >
Movies All >
TV Shows All >
Apps All >


AT&T 11:12 PM
General Restrictions

In-App Purchases ON
Ratings For United States >
Music & Podcasts Explicit >
Movies All >
TV Shows All >
Apps All >

Game Center:

Multiplayer Games ON
Adding Friends ON

FIGURE 1.5 (left) Device application controls on an iPhone

FIGURE 1.6 (middle) Location, accounts, and content controls on an iPhone

FIGURE 1.7 (right) Content and game center controls on an iPhone

iTunes does not offer many ways to limit the iOS device features that a user can access. However, the iOS devices themselves do offer some limits, as we'll see. Remember that this is a manual process that you'll have to repeat on every device . . . manually. In other words, this method is not going to scale well at all. But, again, for a SOHO/small company, it's an easy-to-use, easy-to-understand solution that comes free with every iOS device.

To get to the restriction settings, go into Settings > General > Restrictions. As you can see in the **Figures 1.5**, **1.6**, and **1.7**, you have a *lot* more control over what someone can and cannot do on the device. (The figures are for an iPhone, but the differences between the various devices are so small as to not be worth showing each device's settings separately.)

For most companies, you won't care about most of these settings. (Really, is there a reason to disable Safari?) However, if you want to maintain control of what apps are installed *or* deleted, you can do that here. You can also prevent changes in email accounts, disable camera usage, manage in-app purchases, and disable some of the Game Center features.

Enabling these restrictions requires you to enter a four-number passcode. Assuming you avoid the obvious ones (1234, 3333, and so on), you can set up the restrictions with a fair bit of confidence that they won't be bypassed. Yes, there are ways to bypass these restrictions, and most are not all that difficult. It's almost impossible to lock down a device like this so that it cannot be unlocked. But, for most people, between iTunes and the on-device settings, you should be just fine.

WRAPPING **UP**

Using iTunes and the on-device settings is not a solution you'll want to use for large numbers of iOS devices. But for a small number of devices with simple needs, these controls work quite well. You can simplify the setup process and have some relatively detailed control over what you allow your users to do with their iOS devices. It's not fancy, but it is functional, and that counts.

This page intentionally left blank

2

THE iPHONE CONFIGURATION UTILITY

The iPhone Configuration Utility (iPCU) can be the central point for creating and managing iOS devices for a small company up to a business with hundreds or thousands of devices. This free utility from Apple not only lets you set up applications and provisioning; but with a nice amount of granularity, you can specify *exactly* what a user can and cannot do with his device.

It also offers you more security options than are offered in iTunes or available natively on the device. In this chapter, we'll go over where you can get the iPCU, how you can use it, and the features it offers. In the following chapters, we'll explore a lot of detail on what the iPhone Configuration Utility can do for you and your iOS devices.

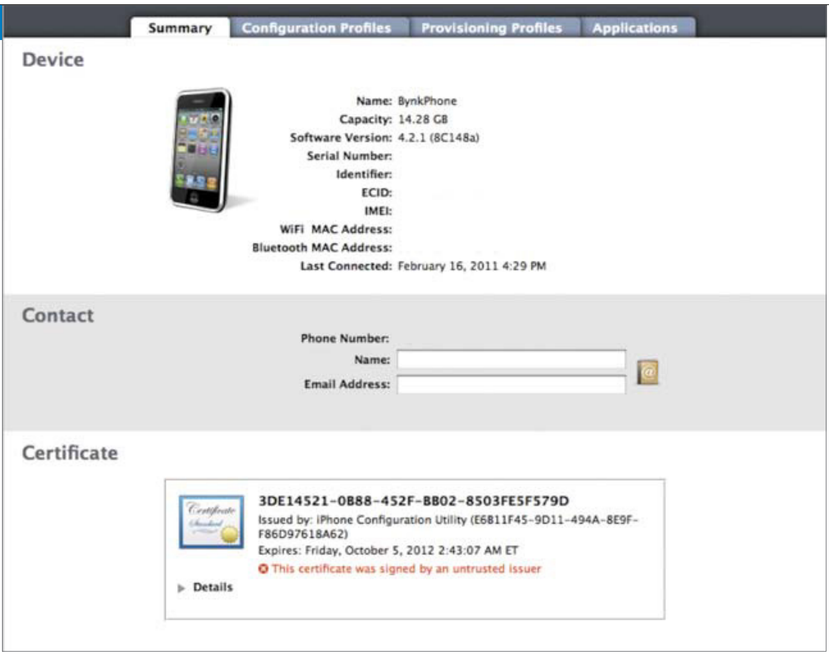
GETTING THE iPCU

The iPCU is available for Mac OS X and Windows from Apple's iPhone Support—Enterprise page at www.apple.com/support/iphone/enterprise/. (Because the specific version of the iPCU can change, that's the best place to find the download links.) With iPCU version 3.2 (the current version at the time of this writing), you need to be running Mac OS X 10.6 or later; or Windows XP SP3, Windows Vista SP1, or Windows 7, and Microsoft .NET Framework 3.5 SP1. Download and install the iPCU version you need, and you're almost ready to start.

APPLE'S iPHONE **BUSINESS PAGES**

I'm going to *highly* recommend that before you start using the iPCU, you spend some time on the iPhone Support—Enterprise page and the iPhone Business Resources page (www.apple.com/iphone/business/resources/). The business resources page, in particular, is a treasure trove of links to useful information for anyone who wants to manage iOS devices and also wants detailed information on exactly how iOS does things. Need details on Exchange, Wi-Fi authentication features, or VPN? It's all there. You will save yourself a great deal of time and troubleshooting by taking a few hours, or days, to read the documentation linked to on the business resources page.

UNDERSTANDING iPHONE CONFIGURATION UTILITY BASICS



Device

Name: BynkPhone
Capacity: 14.28 GB
Software Version: 4.2.1 (8C148a)
Serial Number:
Identifier:
ECID:
IMEI:
WiFi MAC Address:
Bluetooth MAC Address:
Last Connected: February 16, 2011 4:29 PM

Contact

Phone Number:
Name:
Email Address:

Certificate

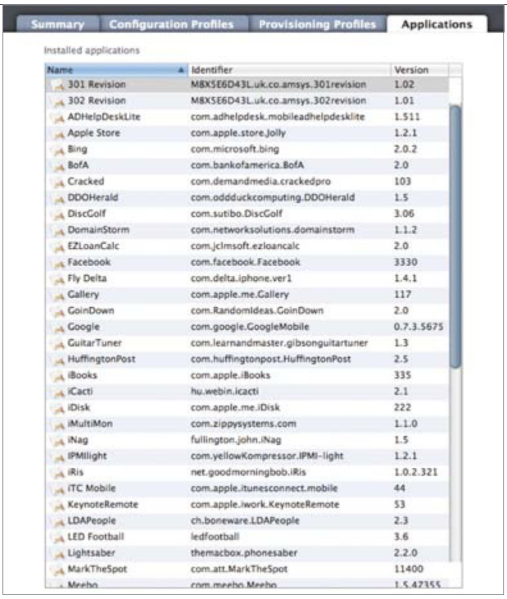
3DE14521-0B88-452F-BB02-8503FE5F579D
Issued by: iPhone Configuration Utility (E6811F45-9D11-494A-8E9F-F86D97618A62)
Expires: Friday, October 5, 2012 2:43:07 AM ET
This certificate was signed by an untrusted issuer
Details

FIGURE 2.1 iPCU summary section

The iPCU has four main configuration sections: Devices, Applications, Provisioning Profiles, and Configuration Profiles.

VIEWING DEVICES

The Devices section is pretty simple: It lists the iOS devices that you've attached via USB to the computer running the iPCU. A summary section shows the basic information for each device—such as OS version, IMEI number, and MAC addresses. (In **Figure 2.1**, some of those numbers are redacted in the images for security/ safety reasons.)



Installed applications

Name	Identifier	Version
301 Revision	M8X5E6D43L.uk.co.amsys.301revision	1.02
302 Revision	M8X5E6D43L.uk.co.amsys.302revision	1.01
ADHelpDeskLite	com.adhelpdesk.mobileadhelpdesklite	1.511
Apple Store	com.apple.store.Jolly	1.2.1
Bing	com.microsoft.bing	2.0.2
BofA	com.bankofamerica.BofA	2.0
Cracked	com.demandmedia.crackedpro	103
DDOHerald	com.oddduckcomputing.DDOHerald	1.5
DiscGolf	com.sutibo.DiscGolf	3.06
DomainStorm	com.networksolutions.domainstorm	1.1.2
EZLoanCalc	com.jclmsoft.ezloancalc	2.0
Facebook	com.facebook.Facebook	3330
Fly Delta	com.delta.iphone.ver1	1.4.1
Gallery	com.apple.me.Gallery	117
GoinDown	com.RandomIdeas.GoinDown	2.0
Google	com.google.GoogleMobile	0.7.3.5675
GuitarTuner	com.learnandmaster.gibsonguitartuner	1.3
HuffingtonPost	com.huffingtonpost.HuffingtonPost	2.5
iBooks	com.apple.iBooks	335
iCacti	hu.webin.icacti	2.1
iDisk	com.apple.me.iDisk	222
iMultiMon	com.zippysystems.com	1.1.0
iNag	fullington.john.iNag	1.5
IPMIlight	com.yellowKompressor.IPMI-light	1.2.1
iRis	net.goodmorningbob.iRis	1.0.2.321
iTC Mobile	com.apple.itunesconnect.mobile	44
KeynoteRemote	com.apple.iwork.KeynoteRemote	53
LDAPeople	ch.boneware.LDAPeople	2.3
LED Football	ledfootball	3.6
Lightsaber	themacbox.phonesaber	2.2.0
MarkTheSpot	com.att.MarkTheSpot	11400
Meebo	com.meebo.Meebo	1.5.47355

FIGURE 2.2 Apps listed in the Applications tab

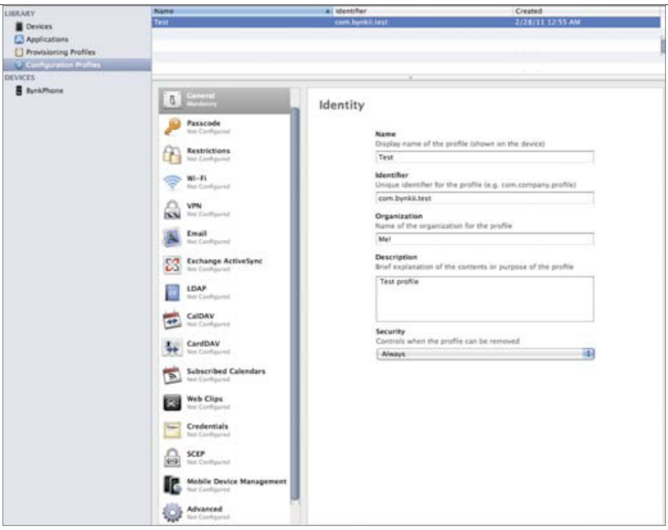
The Configuration Profiles tab shows all configuration profiles used on the device. The Provisioning Profiles tab does the same for provisioning profiles, and the Applications tab (**Figure 2.2**) shows a list of apps installed on the device. As you can see, the apps listing here is functional and not pretty as with iTunes.

There's no real trick to using a device with the iPCU. Open iPCU, connect the device to that computer, and you're ready to go.

USING APPLICATIONS AND PROVISIONING PROFILES

These two are listed together because they go together. "Applications," as used with the iPCU, are not for apps you buy from Apple's App Store. Rather, they're custom apps that your company has written in-house, or commissioned or purchased from a third party. These apps will not normally show up in the App Store, so you can't use that as your distribution mechanism. Instead, you use the iPCU to install these applications on a device.

To distribute applications using the iPCU you need the distribution provisioning profile and the app(s) you want to install. In the Provisioning Profiles tab is where you manage the provisioning profiles, and in the Applications tab you manage the apps you'll install. iPCU has no surprises as far as tab names go.



Name	Identifier	Created
Test	com.bynkii.test	2/28/11 12:55 AM

Identity

Name
Display name of the profile (shown on the device)
Test

Identifier
Unique identifier for the profile (e.g. com.company.profile)
com.bynkii.test

Organization
Name of the organization for the profile
Me!

Description
Brief explanation of the contents or purpose of the profile
Test profile

Security
Controls when the profile can be removed
Always

FIGURE 2.3 Configuration profiles settings

SETTING UP CONFIGURATION PROFILES

This tab gets the most use when you're managing iOS devices (**Figure 2.3**). Here you configure device settings, ranging from installing a standard set of web clips to configuring email accounts, security, and even cellular settings. You'll be spending a lot of time with this tab.

APPLYING PROFILES WITH A CONNECTED DEVICE

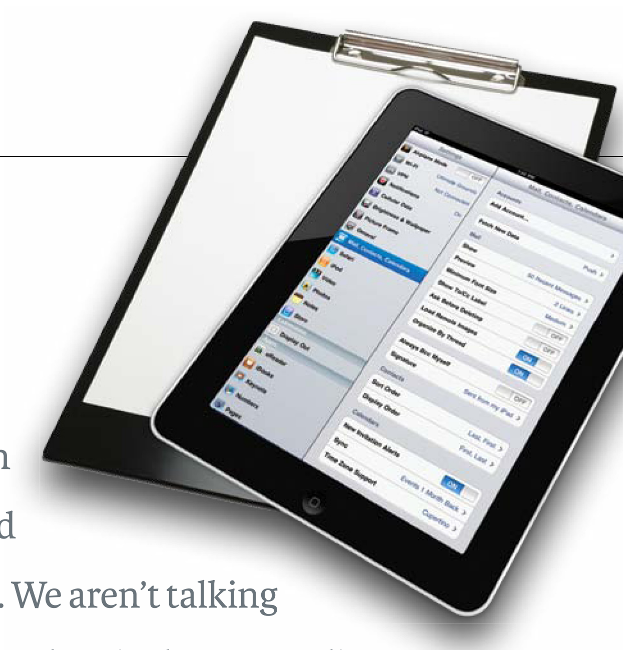
If you have a device attached to your computer while the iPCU is running, a Devices section will appear. In the toolbar, you have buttons to create a new configuration profile, to share your configuration profiles via email and Mail.app, and to export a configuration profile as a .mobileconfig XML file.

WRAPPING **UP**

That's really all there is to the iPCU in terms of major features. Just Devices, Applications, Provisioning Profiles and Configuration Profiles. As we'll see in the next few chapters, that's enough to manage a large number of devices without doing a lot of work.

3

APPS AND PROVISIONING



One of the iPhone Configuration Utility's jobs is to help you install and manage applications on iOS devices. We aren't talking about apps from the App Store, but, rather, in-house applications written for your company that will be used only by company-authorized devices. These are also known as enterprise apps.

Enterprise apps differ from App Store products in a number of ways. First, they aren't vetted or looked at by Apple. There are no rules as to what an enterprise app can or cannot do. They aren't distributed via the App Store, either. In this chapter, you'll see how you can use the iPCU to install enterprise apps on an iOS device. (However, we won't be looking at how you create an app because that's beyond the scope of this book.)

USING **PROVISIONING PROFILES**

A provisioning profile is a binary file used to distribute apps to iOS devices outside of the App Store. It contains certificate, device, and app information required to install an enterprise app on an iOS device.

One important caveat here is that you can't add random provisioning profiles and apps to the iPCU and install an app on any device you choose. App installation requires more than a little coordination between the app's developers and users. Each device that will receive the app must be registered in the company's iOS provisioning portal. Once that is done, a provisioning profile for that device can be generated.

A POINT ABOUT **REGISTRATION**

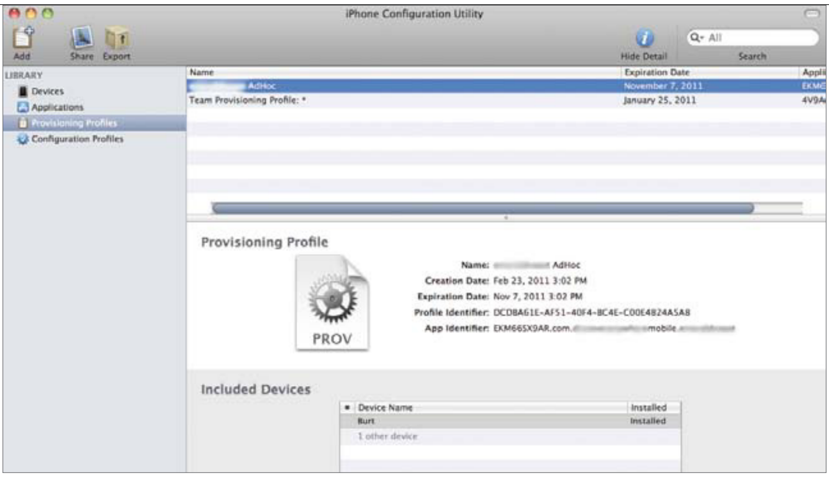
If you are a member of the iOS Developer *Enterprise* Program, you do not have to register your devices with Apple just to distribute apps to them. This situation frequently confuses people. I regularly find myself assuming I have to register devices when I don't. If you *aren't* a member of the iOS Developer Enterprise Program, then you *do* have to register your devices to distribute your own apps to those devices outside of the Apple App Store. If you are a member of the Enterprise program, you only have to register your devices with Apple if you want them to be development devices.

UNDERSTANDING THE PROVISIONING PORTAL

It's important to talk a bit about the provisioning portal because it's central to distributing apps. Unless you jailbreak your iOS devices (a technique that we are not going to discuss in this book), you need a provisioning profile to install iOS apps that aren't on the App Store.

The provisioning portal is key to using apps for two primary reasons: One reason is to set up devices so that you can test apps before submitting them to the App Store—but we don't really care about that for this book. The other reason, which we do care about—is managing provisioning profiles and iOS devices to ensure that you can install your apps on those devices with ease and efficiency.

However, to use the provisioning portal, you must be a registered member of Apple's iOS Developer Program or iOS Enterprise Developer Program.



Name	Expiration Date	Appli
[redacted] AdHoc	November 7, 2011	EKM6
Team Provisioning Profile: *	January 25, 2011	4V9A

Name: [redacted] AdHoc
Creation Date: Feb 23, 2011 3:02 PM
Expiration Date: Nov 7, 2011 3:02 PM
Profile Identifier: DCDBA61E-AF51-40F4-BC4E-C00E4824A5A8
App Identifier: EKM665X9AR.com.[redacted]mobile.[redacted]

*	Device Name	Installed
	Burt	Installed
	1 other device	

FIGURE 3.1 Provisioning Profiles selected in the iPCU

When you're a member, you can log into the iOS Dev Center and use the portal at http://developer.apple.com/ios/manage/overview/index.action. I'm not going to go over the portal in great detail. A lot of it involves things that only developers care about. But because I am going to talk about the provisioning portal throughout this chapter and the rest of the book, I thought it a good idea to spend some time on the portal.

LEARNING MORE ABOUT PROFILES AND DEVICES

While somewhat annoying, using provisioning profiles ensures that you get the application you think you're getting, and *only* that app. It would be a bigger annoyance to discover that the app you installed was modified to include root kits and other malware. By requiring the use of digitally signed apps with a device- or company-specific provisioning profile, the potential for this kind of problem is mitigated.

By the same token, this authentication process also ensures that random file sharing sites can't distribute your company's work to the entire Internet—an important consideration if the app in question contains proprietary or sensitive data.

Once the provisioning profile is created by the developer, using it is dead simple. Open the iPCU, and in the Library, select Provisioning Profiles (**Figure 3.1**). Then, you can either drag the profile (a .mobileprovision file) into the top pane of the Provisioning Profile section; or, click the Add button, navigate to the profile, and click Open to add it to the iPCU. After the profile is installed, you can see it in the top pane of the iPCU's Provisioning Profile section.

PERFORMING **LARGER SCALE** DISTRIBUTION

When you have only a few devices to manage, it's okay to create a unique provisioning profile for each device and manually add it to your company's iOS portal, but what about when you need to distribute applications to large numbers of devices?

In that case, you need to create a file that can add all your company's devices to the portal at once. So let's take a look at doing just that.

UPLOADING MULTIPLE DEVICES

First, you'll need to inform the iPCU about all your devices. That's pretty simple. Just plug the device into the iPCU long enough for it to show up in the Devices section. Once the device is recognized, you can unplug it and move on to the next. The iPCU will remember each device. Then select the devices you want to add to your company's provisioning portal, and choose File > Export. Give the file a name, and the iPCU will create a .deviceids file for all those devices.

The .deviceids file is an XML plist-style file that contains one entry per device. Each entry contains the device's name and identifier, a unique alphanumeric string that identifies the device to the provisioning portal. For example, here are the contents of a .deviceids file with just my iPhone in it (with the device identifiers changed):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.
→apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Device UDIDs</key>
    <array>
        <dict>
            <key>deviceIdentifier</key>
            <string>67d9631117a7a63042100deba49e3990289e6865</string>
            <key>deviceName</key>
            <string>BynkPhone</string>
        </dict>
    </array>
</dict>
</plist>
```

It's pretty simple to figure out. Every additional device has a new `<dict>` section that contains its name and identifier. The handy thing about this is that if you already have those two pieces of information somewhere else—like an asset database of some kind—you could use that database to generate this file without using the iPCU. As you'll see in Chapter 6, you could create a script that would read the data in the iPCU, and then automatically generate the .deviceids file for you.

Or, iPCU could generate a .deviceids file of company devices for use not only with the provisioning portal, but also to *create* a database of device names and identifiers that you can use in other ways.

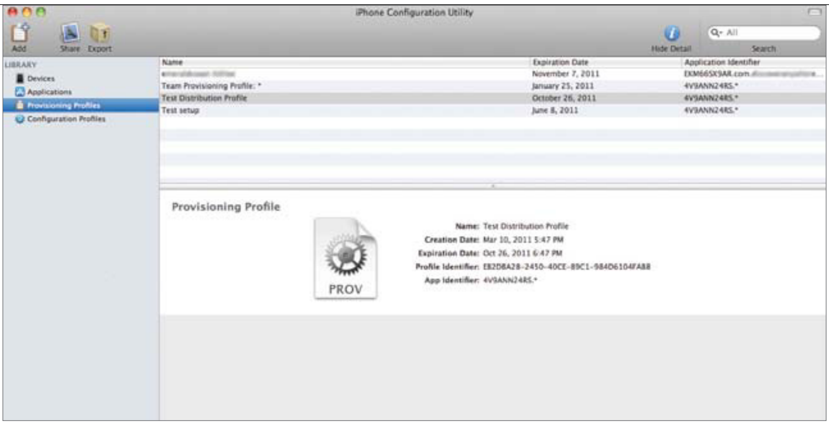
Apple is smart enough to use plain text for this file, so your options for using the .deviceids file are rather large.

However you create the file, you then upload it into your company's provisioning portal. To do that, go to your iOS provisioning portal, and select Devices. In the Manage tab, you'll see two buttons: Upload Devices and Add Devices. Click Add Devices when you want to add just one device at a time. Click Upload Devices to add multiple devices at once via a .deviceids file.

To add one device, click Upload Devices, and then click Choose File. Navigate to your .deviceids file, and click Choose. Then click the Submit button, and the portal will process your devices.

BIG SCARY PORTAL **WARNING**

If you read the fine print, you saw that you can register a specific number of devices each year. It's not *real* small, but it's not real large either, and it depends on your level of developer program membership. However, because you can delete devices, you may think, "I'll just blow out the old ones as needed." But it's not that simple. Removing a device removes it from the portal, but it still counts against your devices-per-year count. So, if you have 99 devices and a maximum count of 100 devices per year, you can remove 40 devices and still only have one device free for that year. Again, neither registration, nor these limits, apply to simply distributing apps when you are a member of the Enterprise program. If you aren't a member of the Enterprise program, then registration and limits do apply.



Name	Expiration Date	Application Identifier
[redacted]	November 7, 2011	EXM66SX9AR.com.[redacted]
Team Provisioning Profile: *	January 25, 2011	4V9ANN24RS.*
Test Distribution Profile	October 26, 2011	4V9ANN24RS.*
Test setup	June 8, 2011	4V9ANN24RS.*

Provisioning Profile

PROV

Name:	Test Distribution Profile
Creation Date:	Mar 10, 2011 5:47 PM
Expiration Date:	Oct 26, 2011 6:47 PM
Profile Identifier:	EB2D8A28-2450-40CE-89C1-984D6104FA88
App Identifier:	4V9ANN24RS.*

FIGURE 3.2 Distribution profile added to the iPCU

Now you're set to create a single enterprise distribution profile that will work for all your devices.

APPLYING DISTRIBUTION PROFILES

As a warning, this section assumes that you have done all the necessary work to set up your provisioning portal. At that point, creating a distribution profile is pretty simple.

Go to the Distribution section, and in the Prepare App tab, click "Create and Download a Distribution Provisioning Profile for Enterprise In-House Distribution." You'll see step-by-step instructions for creating your enterprise distribution profile. Once the profile is created, you can download it by going to the Provisioning section of the provisioning portal, selecting the Distribution tab, and clicking Download.

When the file is downloaded, drag it to the top pane of the Provisioning Profile section in the iPCU (**Figure 3.2**) and you're set.

Really, this looks a lot more complicated in text than it does in real life. If you read the documentation from Apple for the portal and pretty much do as it tells you, you can have an enterprise distribution profile ready to go in a few minutes.

However you choose to proceed, you now have your provisioning profile set up in the iPCU. Now let's set up your app.

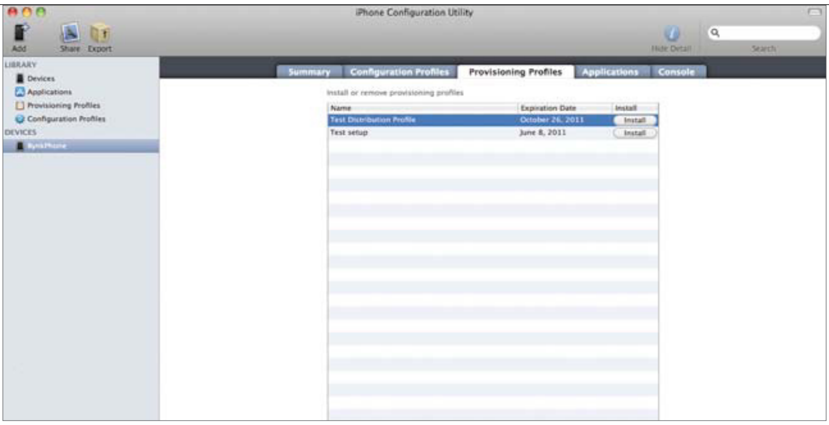
USING **APPLICATIONS**

This is actually the easiest part. First, you need the app you're going to install. Note that you don't actually need a provisioning profile to set up an app in the iPCU. Of course, you can't *install* the application without that profile, so setting up the app without the profile is of little use. But it's worth noting that the iPCU doesn't care about the order of setup. Once you have the app file in the iPCU, you can click Applications and drag the app file into the top pane of the Applications section; or you can click Add, navigate to the app file, and click Open. Either way, once you're done, your app is ready to go. Really, it's that simple. Get the app and the provisioning profile set up in the iPCU and you're ready to install the app.

INSTALLING AND UNINSTALLING APPS AND PROFILES

You have your provisioning profile and your app. Now, it's time to install. When using the iPCU, installing an app onto a device is not much harder than installing an app to the iPCU as long as you remember the proper order: profile first, then app. Actually, you can't really get that wrong because the iPCU won't let you install an app on a device that doesn't have the correct profile.

First, fire up the iPCU and attach a device to it via USB. Obviously, this should be a device with a profile that you created. The iPCU will sprout a Devices section with the connected device displayed in it. This section is different than the *other* Devices section that is always there. *This* Devices section is where you manage a currently-connected device.



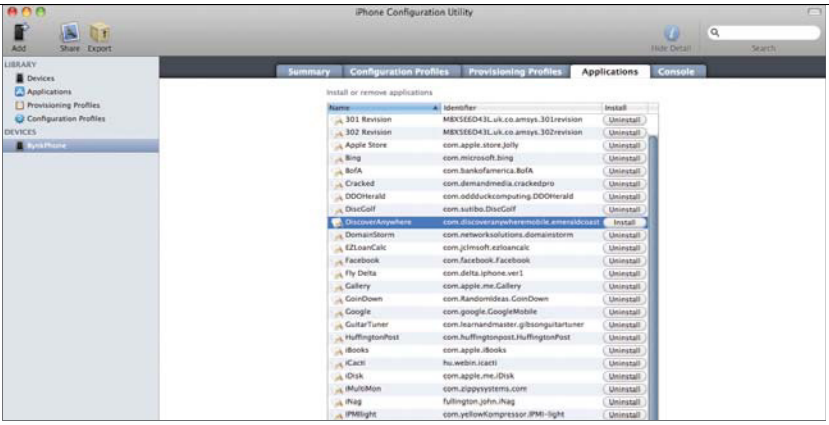
Install or remove provisioning profiles

Name	Expiration Date	Install
Test Distribution Profile	October 26, 2011	Install
Test setup	June 8, 2011	Install

FIGURE 3.3 Profiles ready for installation to a device

When you select the currently-connected device in the iPCU, you'll see a row of tabs across the top of the pane. The first tab you're concerned with is Provisioning Profiles. Click it and you'll see all the profiles that may be installed on the device (**Figure 3.3**). Each of those profiles has an Install button. Click Install for the desired profile.

Once that's done (it takes about two seconds for an iPhone 3GS), the button changes to a Remove button. (You can click Remove to manually remove the profile from the device. Amusingly, if you remove the profile, you get a Big Scary Warning that this action cannot be undone. Well, in the Command-Z sense of undo, I suppose that's true. However, if you click the Install button again, you'll reinstall the profile and undo the removal.)



Install or remove applications

Name	Identifier	Install
301 Revision	MBXSEED43L.uk.co.amsys.301revision	Uninstall
302 Revision	MBXSEED43L.uk.co.amsys.302revision	Uninstall
Apple Store	com.apple.store.Jolly	Uninstall
Bing	com.microsoft.bing	Uninstall
BofA	com.bankofamerica.BofA	Uninstall
Cracked	com.demandmedia.crackedpro	Uninstall
DDOHerald	com.oddduckcomputing.DDOHerald	Uninstall
DiscGolf	com.sutibo.DiscGolf	Uninstall
DiscoverAnywhere	com.discoveranywheremobile.emeraldcoast	Install
DomainStorm	com.networksolutions.domainstorm	Uninstall
EZLoanCalc	com.jclmsoft.ezloancalc	Uninstall
Facebook	com.facebook.Facebook	Uninstall
Fly Delta	com.delta.iphone.ver1	Uninstall
Gallery	com.apple.me.Gallery	Uninstall
CoinDown	com.Randomideas.CoinDown	Uninstall
Google	com.google.GoogleMobile	Uninstall
GuitarTuner	com.learnandmaster.gibsonguitartuner	Uninstall
HuffingtonPost	com.huffingtonpost.HuffingtonPost	Uninstall
iBooks	com.apple.iBooks	Uninstall
iCacti	hu.webin.icacti	Uninstall
iDisk	com.apple.me.iDisk	Uninstall
iMultiMon	com.zippysystems.com	Uninstall
iNag	fullington.john.iNag	Uninstall
IPMIlight	com.yellowKompressor.IPMI-light	Uninstall

FIGURE 3.4 The Applications tab lists all apps installed on the device.

The next tab you care about is the Applications tab (**Figure 3.4**). Again, you'll find no surprises here. When you click the Applications tab, you'll see all the enterprise or App Store apps installed on this device.

What you're looking for are apps with an Install button next to them. Assuming you've installed the right profile, click Install for the desired app and a few seconds later, it's installed. No fuss, no muss.

Currently installed apps will have an Uninstall button next to them. If you click Uninstall for an App Store app, you'll get another Big Scary Warning about not being able to undo your action. This time, however, it's not kidding. If you want to reinstall an App Store app, it's back to iTunes you go.

On the other hand, if you want to remove App Store apps that shouldn't be there, the iPCU is a *very* fast way to do so. Click Uninstall, click OK to the warning, and in a few seconds the app is gone—no requests for final reviews, no nothing. Bang. Gone. It's not deleted out of iTunes, but it is definitely off the device.

WRAPPING **UP**

This was a busy chapter, especially in terms of profiles and portals. But, understanding them will be important when you later learn about other ways to distribute apps, including *wireless distribution*. For now, you should be able to install apps via the iPCU and understand the process behind that installation. The installation mechanics may change in terms of step-by-step activity, but the principles will probably stay constant.

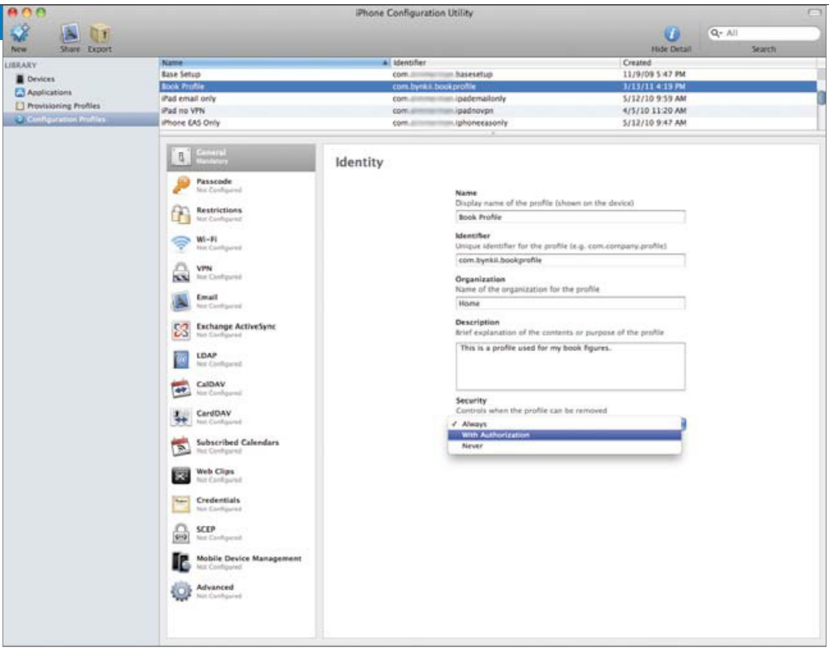
4

CREATING **CONFIGURATION PROFILES**

This is the "big" chapter for the iPhone Configuration Utility (iPCU), and for anyone who needs to perform extensive/advanced management of iOS devices. Configuration profiles are XML files that control the behavior of your iOS devices, restrict or allow specific features, and set up functions such as email and calendars. Once created, configuration profiles can be distributed via USB, email, or a web page.

You can also encrypt and sign a profile to restrict the devices it can be applied to, and password-protect it so that it cannot be removed short of wiping all the data on the device. Configuration profiles can be long and complicated, or kept simple and used only to point a device at a Mobile Device Management server that can push the desired settings to the devices.

USING **GENERAL SETTINGS**



Name	Identifier	Created
Base Setup	com.[redacted].basesetup	11/9/09 5:47 PM
Book Profile	com.bynkii.bookprofile	3/13/11 4:19 PM
iPad email only	com.[redacted].ipademailonly	5/12/10 9:59 AM
iPad no VPN	com.[redacted].ipadnovpn	4/5/10 11:20 AM
iPhone EAS Only	com.[redacted].iphoneeasonly	5/12/10 9:47 AM

Identity

Name
Display name of the profile (shown on the device)
Book Profile

Identifier
Unique identifier for the profile (e.g. com.company.profile)
com.bynkii.bookprofile

Organization
Name of the organization for the profile
Home

Description
Brief explanation of the contents or purpose of the profile
This is a profile used for my book figures.

Security
Controls when the profile can be removed
✓ Always
With Authorization
Never

FIGURE 4.1 General settings for identity security

When you're creating a configuration profile, the iPCU might have you thinking that the General settings are the only mandatory settings. That's not exactly correct. You need General plus at least one additional setting. This is logical because it doesn't make sense to have a configuration that has only the General section because the General section doesn't really *do* anything in terms of configuring your iOS device. It's just there to identify the profile.

But, if you aren't expecting that behavior, and you're testing a profile, it could be annoying when the device refuses to install a profile with only the General settings configured.

The General settings are concerned only with the profile identity security (**Figure 4.1**). As such, the number of settings here is pretty small. You can set the name of the profile, which is what users see when they go into the General settings on the device and select Profiles (**Figure 4.2**). That's not a misprint by the way; you can have multiple profiles on a single device. If you choose to have multiple profiles, however, please watch your settings. Choosing conflicting settings would be . . . bad.

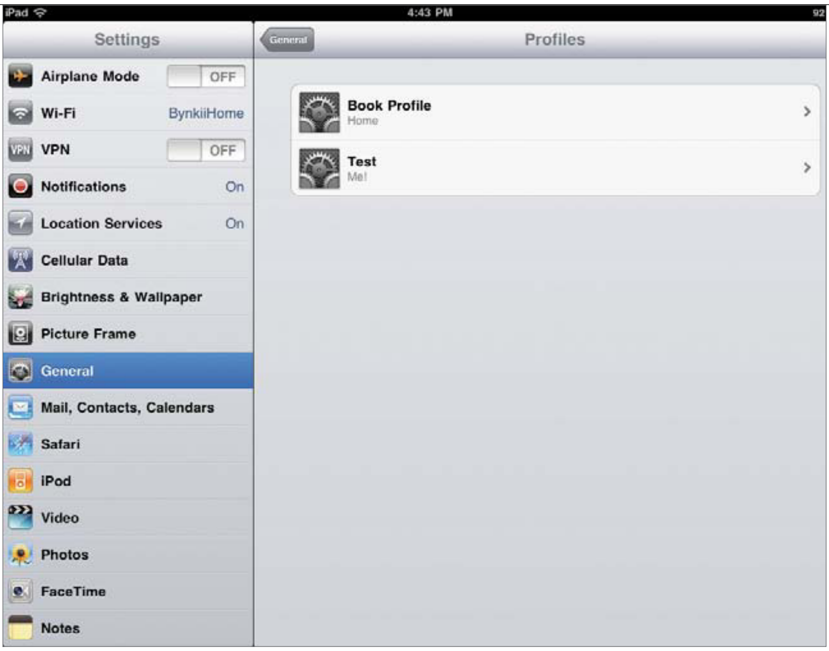
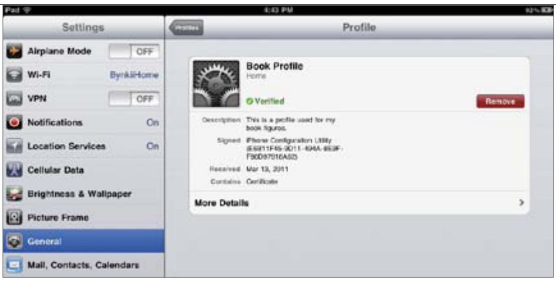


FIGURE 4.2 General settings on an iOS device

Most of the settings here are descriptive, starting with a Name for the profile that the user will see on the device and a unique Identifier named similarly to plist files—for example, "com.bynkii.bookprofile"—that serves two important purposes: First, if no other profile on a device has that same identifier, the profile's settings are added to the device. Second, if a profile on the device does have the same identifier, the settings in the new profile replace the previous profile's settings.

This functionality can make it easy to update an existing profile. Rather than performing a full remove and replace, you can just edit an existing profile while retaining the same identifier, and re-install the profile on the device. The exception is when changing Exchange accounts. When changing an Exchange account, you must remove the profile with the Exchange info so that the Exchange data can be purged.

The Organization and Description fields are available for you to insert customized information that—like the Name field contents—are displayed on the device.


Settings

Airplane Mode OFF

Wi-Fi BynkiiHome

VPN OFF

Notifications On

Location Services On

Cellular Data

Brightness & Wallpaper

Picture Frame

General

Mail, Contacts, Calendars

Profiles

Profile

Book Profile

Home

Verified

Remove

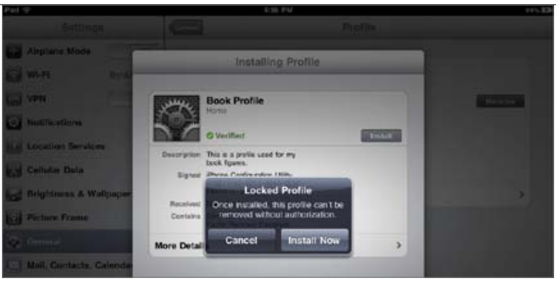
Description This is a profile used for my book figures.

Signed iPhone Configuration Utility (E6911F45-9D11-494A-853F-F90D97016A52)

Received Mar 13, 2011

Contains Certificate

More Details


Installing Profile

Book Profile

Home

Verified

Install

Description This is a profile used for my book figures.

Locked Profile

Once installed, this profile can't be removed without authorization.

Cancel

Install Now

FIGURE 4.3 (left) Click Remove to delete a profile from a device.

FIGURE 4.4 (right) Warning displayed when installing an app that requires authorization

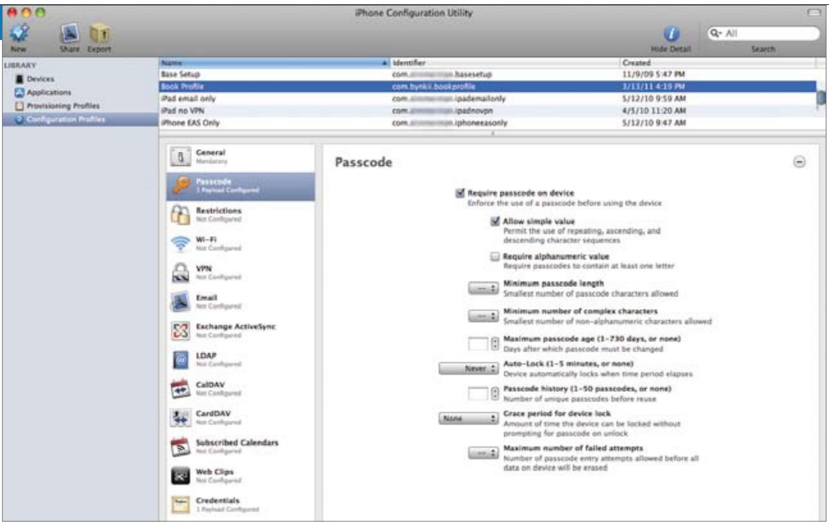
The Security setting controls the ability to remove the profile and has three settings: Always, With Authorization, and Never. Always is pretty self-explanatory: The profile can always be removed by clicking Remove in the profile's information section (**Figure 4.3**).

If you choose With Authorization, the profile can be removed only by entering a passcode that you set when you create the profile in the iPCU. You'll see a warning to that effect when you install a profile with this security setting (**Figure 4.4**).

If you choose the final option, Never, you're required to erase all the data from the device to remove the profile.

Realistically, don't choose Never unless you are in a high-security environment (and I mean "Lawrence Livermore National Laboratory where we perform nuclear weapons research," not just "I don't want people to know about our new website data" security); or unless you are not going to update that profile anytime soon. Choosing Never usually causes far more problems than it solves. Choosing With Authorization will cover you 99 percent of the time. If the device is a personal device that someone is using for work purposes, consider choosing Always. because it's kind of rude to lock someone out of her own phone.

SETTING A **PASSCODE**



Name	Identifier	Created
Base Setup	com.[redacted].basesetup	11/9/09 5:47 PM
Book Profile	com.bynkii.bookprofile	3/13/11 4:19 PM
iPad email only	com.[redacted].ipademailonly	5/12/10 9:59 AM
iPad no VPN	com.[redacted].ipadnovpn	4/5/10 11:20 AM
iPhone EAS Only	com.[redacted].iphoneeasonly	5/12/10 9:47 AM

Passcode

[✓] Require passcode on device
Enforce the use of a passcode before using the device

[✓] Allow simple value
Permit the use of repeating, ascending, and descending character sequences

[ ] Require alphanumeric value
Require passcodes to contain at least one letter

-- Minimum passcode length
Smallest number of passcode characters allowed

-- Minimum number of complex characters
Smallest number of non-alphanumeric characters allowed

Maximum passcode age (1–730 days, or none)
Days after which passcode must be changed

Never Auto-Lock (1–5 minutes, or none)
Device automatically locks when time period elapses

Passcode history (1–50 passcodes, or none)
Number of unique passcodes before reuse

None Grace period for device lock
Amount of time the device can be locked without prompting for passcode on unlock

-- Maximum number of failed attempts
Number of passcode entry attempts allowed before all data on device will be erased

FIGURE 4.5 Passcode settings in iPCU

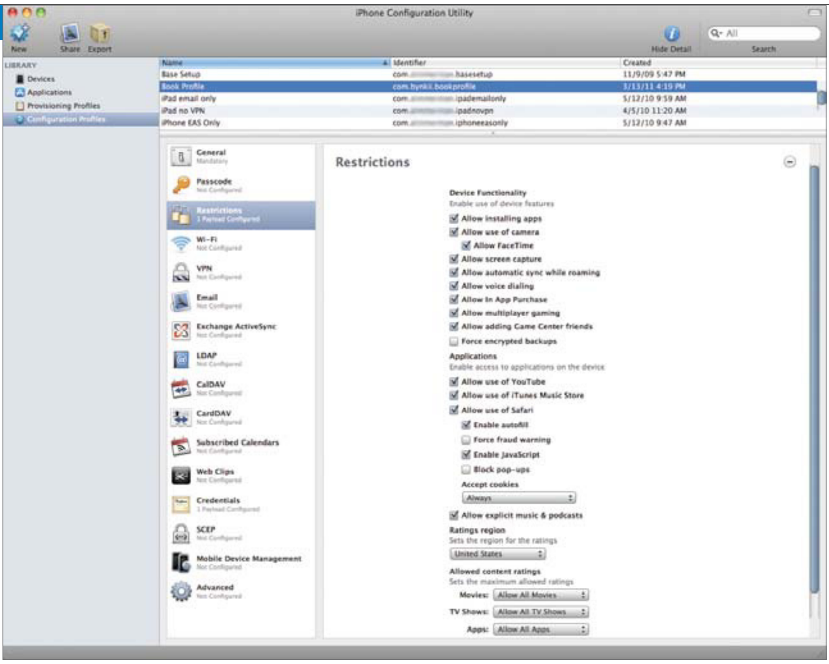
Next are the Passcode settings (**Figure 4.5**). These settings are simple enough, but you have a *lot* more granularity than you can access using only the device or iTunes.

For example, if you don't want people to choose a passcode like "7777", simply deselect "Allow simple value." If you want to force the use of letters and numbers in a passcode, select "Require alphanumeric value." You can set the "Minimum passcode length" from 1 to 16 characters, set the "Minimum number of complex characters" (such as &, $, or !) from 1 to 16, and choose "Maximum passcode age" up to 730 days. You can also restrict the number of unique passcodes that must be used before a passcode can be repeated, along with a few other passcode settings.

Really, none of the settings available differ much from password settings of any standard IT setup, save one: "Maximum number of failed attempts." This setting can be dangerous, because it lets you limit the number of times an incorrect passcode can be entered *before the device erases itself* (up to a maximum of 16 tries.) Yes, this restriction can be useful if your iOS devices carry critical confidential data; but, if this value is set too low, you can create a large user support headache for yourself.

This setting falls into a category I like to call "If you have to ask, the answer is No!" so, if you are currently asking yourself, "Should I use this?", then don't use it. If you actually need to use this setting, you'll know and you won't have to ask. Any setting that can cause a device to erase itself should be implemented with the greatest care and caution.

CHOOSING **RESTRICTIONS**



Name	Identifier	Created
Base Setup	com.[illegible].hasesetup	11/9/09 5:47 PM
Book Profile	com.bynkii.bookprofile	3/13/11 4:19 PM
iPad email only	com.[illegible].ipademailonly	5/12/10 9:59 AM
iPad no VPN	com.[illegible].ipadnovpn	4/5/10 11:20 AM
iPhone EAS Only	com.[illegible].iphoneeasonly	5/12/10 9:47 AM

General	Mandatory
Passcode	Not Configured
Restrictions	1 Payload Configured
Wi-Fi	Not Configured
VPN	Not Configured
Email	Not Configured
Exchange ActiveSync	Not Configured
LDAP	Not Configured
CalDAV	Not Configured
CardDAV	Not Configured
Subscribed Calendars	Not Configured
Web Clips	Not Configured
Credentials	1 Payload Configured
SCEP	Not Configured
Mobile Device Management	Not Configured
Advanced	Not Configured

Restrictions

Device Functionality
Enable use of device features
- ✓ Allow installing apps
- ✓ Allow use of camera
  - ✓ Allow FaceTime
- ✓ Allow screen capture
- ✓ Allow automatic sync while roaming
- ✓ Allow voice dialing
- ✓ Allow In App Purchase
- ✓ Allow multiplayer gaming
- ✓ Allow adding Game Center friends
- ☐ Force encrypted backups

Applications
Enable access to applications on the device
- ✓ Allow use of YouTube
- ✓ Allow use of iTunes Music Store
- ✓ Allow use of Safari
  - ✓ Enable autofill
  - ☐ Force fraud warning
  - ✓ Enable JavaScript
  - ☐ Block pop-ups
  - Accept cookies: Always
- ✓ Allow explicit music & podcasts

Ratings region
Sets the region for the ratings
United States

Allowed content ratings
Sets the maximum allowed ratings
Movies: Allow All Movies
TV Shows: Allow All TV Shows
Apps: Allow All Apps

FIGURE 4.6 Restrictions settings

Restrictions is the most complex group of settings and the worst named. Restrictions. Really? This is the best Apple could come up with? Anyway, these settings enable and disable most of the basic hardware functionality on the device (**Figure 4.6**).

I'm not going to detail every one of these settings, as there's a ton of them and they're all pretty self-explanatory. However, a few are worth noting. For example, if you want to avoid the *hideous* roaming charges that both AT&T and Verizon love so well (in the U.S. at least), deselect "Allow automatic sync while roaming." A user can still check her communications automatically, and a device that's not being actively used won't rack up a few thousand dollars of data charges because that user gets a lot of email.

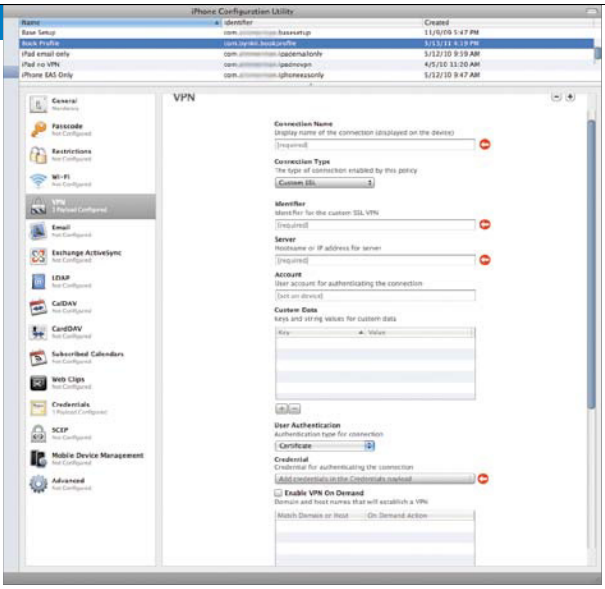
Selecting "Force encrypted backups" just makes sense, and causes almost no noticeable problems on the user's end. You may also want to deselect "Allow explicit music & podcasts" if the device is mostly used in a corporate setting. "Allow use of iTunes Music Store" applies only to the device. It's not going to stop people from using iTunes on their Mac or Windows computers.

CONFIGURING **WI-FI**

The Wi-Fi settings are short and to the point. You specify the SSID of the network, whether or not it's hidden, its security type, and the password (for personal networks). If you specify Enterprise in the "Security Type" pop-up menu, you reveal many more options, such as several Extensible Authentication Protocol (EAP) types; Authentication information; and Trust, which contains the certificates used to validate the authentication server for the network.

Notice that you can set up multiple networks here. So, if a user must travel between facilities— each with its own Wi-Fi network setup—you can populate his iOS device with the information for each network ahead of time, so his device just works. This also saves you from handing everyone the password to the wireless network, which is not only a good idea, but sometimes a regulatory requirement.

SETTING UP **VPN**



Name	Identifier	Created
Base Setup	com.[redacted].basesetup	11/9/09 5:47 PM
Book Profile	com.bynkii.bookprofile	3/13/11 4:19 PM
iPad email only	com.[redacted].ipademailonly	5/12/10 9:59 AM
iPad no VPN	com.[redacted].ipadnovpn	4/5/10 11:20 AM
iPhone EAS Only	com.[redacted].iphoneeasonly	5/12/10 9:47 AM

VPN

Connection Name
Display name of the connection (displayed on the device)
[required]

Connection Type
The type of connection enabled by this policy
Custom SSL

Identifier
Identifier for the custom SSL VPN
[required]

Server
Hostname or IP address for server
[required]

Account
User account for authenticating the connection
[set on device]

Custom Data
Keys and string values for custom data

Key	Value

User Authentication
Authentication type for connection
Certificate

Credential
Credential for authenticating the connection
Add credentials in the Credentials payload

☐ **Enable VPN On Demand**
Domain and host names that will establish a VPN

Match Domain or Host	On Demand Action

FIGURE 4.7 VPN settings are easy to configure incorrectly.

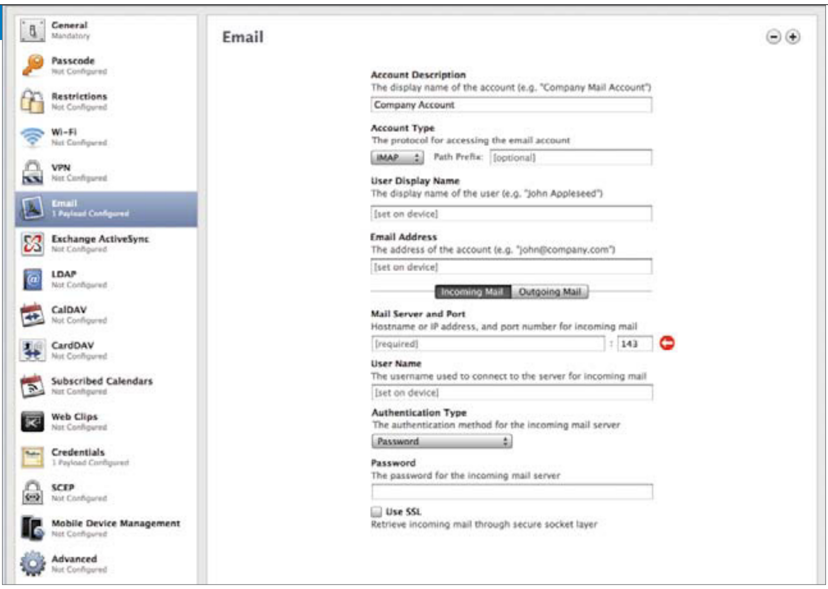
The VPN settings are the opposite of the Wi-Fi settings: They are not to the point, not terribly clear, and painfully easy to incorrectly configure so that your VPN will never work (**Figure 4.7**).

In addition, some of the VPN types listed in the iPCU—such as F5 SSL, Cisco AnyConnect, and Juniper SSL—require you to first install the appropriate apps from the Apple App Store. Not fun.

I'm not trying to downplay the importance of VPN or to avoid it; but there are six specified VPN connection types along with a "Custom SSL" setting, and at least 108 separate VPN settings; and, as with the Wi-Fi settings, you can configure *multiple* VPNs setups here. Furthermore, if you are dealing with "VPN on Demand" and some of the custom options, you'll be looking at *even more* settings.

Fully explaining all of that is literally a book unto itself and, in fact, many VPN books have been written. VPNs are still overly complex as the VPN settings show. If you are responsible for maintaining your VPN, I don't need to tell you what settings to use. If you are not the person maintaining your VPN, schedule some time to sit down with your VPN guru, the iPCU, and an iOS device to work out the best settings for your devices. Then use them. Take copious notes—you'll be glad you did.

SETTING UP **EMAIL**



Email

Account Description
The display name of the account (e.g. "Company Mail Account")
Company Account

Account Type
The protocol for accessing the email account
IMAP   Path Prefix: [optional]

User Display Name
The display name of the user (e.g. "John Appleseed")
[set on device]

Email Address
The address of the account (e.g. "john@company.com")
[set on device]

Mail Server and Port
Hostname or IP address, and port number for incoming mail
[required] : 143

User Name
The username used to connect to the server for incoming mail
[set on device]

Authentication Type
The authentication method for the incoming mail server
Password

Password
The password for the incoming mail server

☐ **Use SSL**
Retrieve incoming mail through secure socket layer

FIGURE 4.8 POP/IMAP email account settings

These settings configure POP/IMAP email accounts for your devices (**Figure 4.8**).

One thing right away: Don't use POP. If someone is checking her email with an iOS device, it is fairly certain that it isn't the only way she checks email. You really, really want to use IMAP here. Yes, I know, POP has that "leave it on the server" option. That's nice but it's not IMAP, which is designed from the ground up to be used by multiple devices checking the same account. If you were entering the 24 hours of Le Mans, would you drive an Audi R15 TDI, or a '78 Pinto with a big spoiler and glasspacks? Right. Use IMAP.

By the same token, also use SSL. If you haven't enabled this on your server, now is a great time to do so. Open wireless networks are a gold mine for people doing illicit data mining, and SSL is an easily-implemented method to prevent them from mining you. So, use SSL. The only other thing I'd specifically recommend is using authenticated SMTP. It's a cheap way to stop unauthorized people from using your SMTP server, and iOS supports it.

As far as the other settings go, email settings are email settings are email settings. However, you will want to leave some settings blank for the user to fill in: the User Display Name, the User Name, and the Password. If you fill those out, you'll have to create a custom profile for everyone, or change the settings every time you install them on a device. You don't want to do that, do you? Of course not.

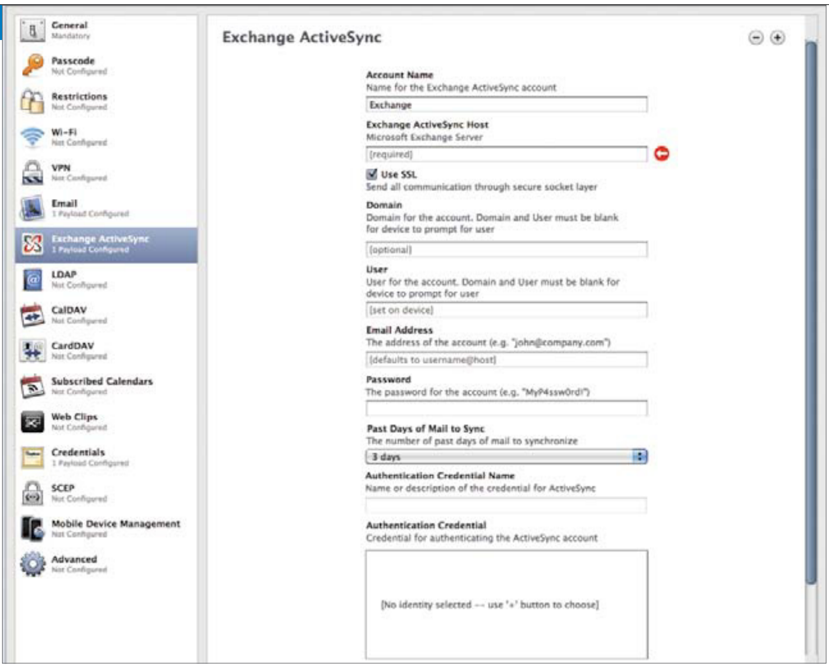
However, the downside of letting users enter that information is if you install a profile via USB, the user has to enter that information during that installation. So, when you're installing profiles to set up email or any other option that requires user input, you won't want to do so over USB. Luckily, as you'll see, you have other ways to distribute these profiles.

The only setting I want to talk about specifically is Path Prefix, because it is a setting you'll use with a rather popular email provider, namely Gmail.

When you set up a Gmail account on an iOS device, you're really setting up an IMAP account—and Google has an "interesting" take on IMAP. If you use the "standard" IMAP settings for Gmail, you'll realize that you have duplicates of every folder in your Gmail account. That's because if you look at your account from the folder root, you'll see you have a [Gmail] tree with all your Gmail "folders" and possibly an [IMAP] tree as well. By setting the Path Prefix to [Gmail], you avoid some of that duplication.

Boy, that's kind of a pain, isn't it? Wouldn't it be nice if you had an easier way to deal with Gmail that set up Google Calendars and Contacts at the same time? Well, you do. You can use Exchange ActiveSync instead of Email for this. Your setup is not only simpler, but you get all your calendaring and contacts, too. (Unfortunately, you still have the same ugly folders issue because Google's Exchange ActiveSync implementation is as "interesting" as their IMAP implementation, but at least you do less work for it.)

USING **EXCHANGE ACTIVESYNC**



Exchange ActiveSync

Account Name
Name for the Exchange ActiveSync account
Exchange

Exchange ActiveSync Host
Microsoft Exchange Server
[required]

☑ **Use SSL**
Send all communication through secure socket layer

Domain
Domain for the account. Domain and User must be blank for device to prompt for user
[optional]

User
User for the account. Domain and User must be blank for device to prompt for user
[set on device]

Email Address
The address of the account (e.g. "john@company.com")
[defaults to username@host]

Password
The password for the account (e.g. "MyP4ssw0rd!")

Past Days of Mail to Sync
The number of past days of mail to synchronize
3 days

Authentication Credential Name
Name or description of the credential for ActiveSync

Authentication Credential
Credential for authenticating the ActiveSync account
[No identity selected -- use '+' button to choose]

FIGURE 4.9 Exchange ActiveSync settings

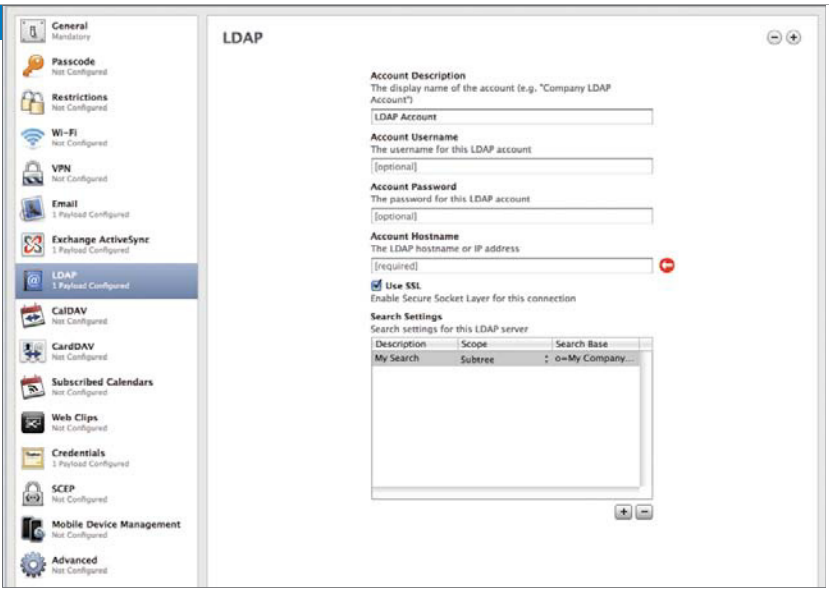
Microsoft Exchange is, for better or worse, the 800-pound gorilla of corporate email; and its mobile protocol, Exchange ActiveSync (EAS), is the way you connect mobile devices to Exchange. However, a while ago, Microsoft did something quite brilliant with EAS: they decoupled it from Exchange. So, it still has the name, but you can use it with servers that have nothing to do with Exchange, such as servers from Google, Kerio, Zimbra, Atmail, and others.

Exchange ActiveSync, name aside, is a boon to iOS devices and the like because—rather than configuring your email server *and* your calendar server *and* your contact server separately—you just set up EAS and you're done. Exchange ActiveSync also allows handy things for IT folk such as the remote wipe of a device without signing up each and every device for "Find my iPad/iPhone." In Exchange ActiveSync settings (**Figure 4.9**), you configure Exchange ActiveSync accounts on your iOS devices.

As with email, you'll find the "normal" fields here, such as Account Name, the server name, Use SSL (YES), and so on. A few fields, however, may appear a bit odd to the uninitiated. First of these is the Domain field. If you are on a Windows network using an Exchange Server, the domain is usually your Windows domain. If you're using a server such as Gmail, or Kerio Connect (that is, something other than an actual Microsoft Exchange server), you can leave this field blank.

The other strange fields are the authentication credential fields. These are used with certificates that validate the *server* to the *client*. This may seem odd, but it is a good way to avoid accidental connection to a rogue Exchange server that happens to have the same DNS address as the desired server. As with email settings, leave the User, Email, and Password fields blank unless you specifically want to fill them in for each user.

ENABLING **LDAP**



LDAP

Account Description
The display name of the account (e.g. "Company LDAP Account")
LDAP Account

Account Username
The username for this LDAP account
[optional]

Account Password
The password for this LDAP account
[optional]

Account Hostname
The LDAP hostname or IP address
[required]

[✓] **Use SSL**
Enable Secure Socket Layer for this connection

Search Settings
Search settings for this LDAP server

Description	Scope	Search Base
My Search	Subtree	o=My Company...

FIGURE 4.10 LDAP settings

Lightweight Directory Access Protocol (LDAP) started as a way to track email contact information and eventually morphed into the way companies manage all their computers and users. While Active Directory, Open Directory, and OpenLDAP are all based on LDAP, for the purposes of this book, LDAP is used only as a contact database.

Most of the settings here are pretty basic (**Figure 4.10**), with a couple of exceptions. The first exception is the "Account Username" field. Depending on your server type, your account name can look like:

- john
- john@company.com
- cn=john,cn=users,dc=example,dc=com

The last example is a full *distinguished name* and completely specifies the user, who in this case is *john*, in the *users* container in the domain *example.com*. It's simple, but only if you know how to read LDAP-ese. In that sense, LDAP is kind of like VPN. If you're running your LDAP server, you already know this; if you aren't, schedule some time with your directory administrator and have that person help you set up this information.

The second LDAP setting that might catch you off guard is the Search Settings field, which requires some basic knowledge of LDAP structure. LDAP is, in general, structured like a tree. At the root, you have the domain, so *company.com* would map to "dc=company,dc=com". Everything expands out from there in a variety of containers (generally, although somewhat incorrectly, abbreviated as CN) and organizational units (OU).

When searching an LDAP directory, you obviously want to limit how much data has to be searched for the benefit of the search speed, and also to control the overall load on the system. A hundred devices searching thousands of entries are going to create a greater server load than if they're searching a few hundred entries. So, you limit searches by the scope, or range, of the search and by the starting point. To set your search scope, you have three options:

- Base, which searches only the defined search base. If the data you want is one level below that base, the search won't find it.
- One Level, which searches the base and the level immediately below it
- Subtree, which searches the base and everything below the base regardless of the number of levels.

The next step is to set up your Search Base or starting point. Since you're really just using LDAP for email contact information, you'll want to set up the container or OU for your users. If your LDAP directory is Apple's Open Directory, typically the OU will be the users container, the Open Directory Master computer name, and the domain name. So:

```
cn=users,dc=odmasterserver,dc=domain,dc=com
```

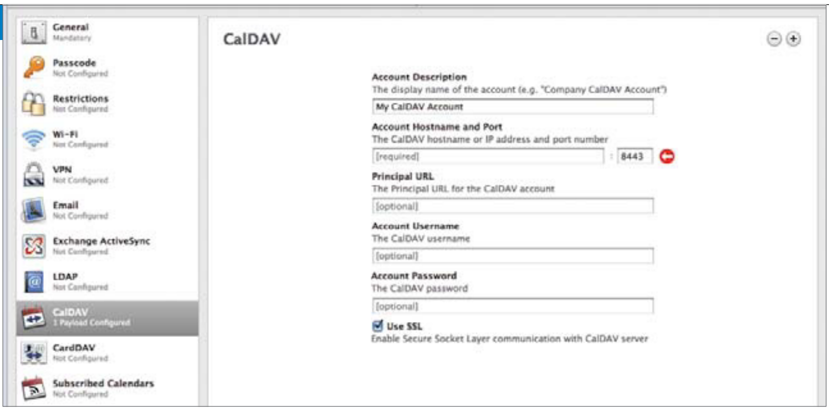
would be a typical search base for a generic Open Directory setup. Other LDAP implementations would be different, but similar to this.

By restricting your starting point to just the users container, you can set the search scope to be "Subtree", and not worry about every device iterating through every part of the LDAP directory just to find Bob's email address. When LDAP is set up, both the Email and Contacts apps on the device will automatically use LDAP data for activities such as autofilling email addresses in messages.

BIG SCARY **LDAP WARNING**

As I just said, LDAP isn't only for email addresses. A modern LDAP implementation can, and often does, contain almost every bit of data about the people and computers in a company. It can also contain office lists, conference room lists, personal phone numbers and addresses, and all sorts of other information that you don't want the world to be able to find. If you are going to set up LDAP on your iOS devices, it is *critical* that you do not allow anonymous access over unencrypted connections. You really, really, *really* want to require SSL, and use both usernames and passwords for everyone allowed to access your LDAP information from outside your company's network. Plus, it's a very good idea to also require it inside the company network. If everything is encrypted, someone who manages to get past your firewall still ends up with a whole lot of nothing for the effort.

SETTING THE DATE WITH **CALDAV**



CalDAV

Account Description
The display name of the account (e.g. "Company CalDAV Account")
My CalDAV Account

Account Hostname and Port
The CalDAV hostname or IP address and port number
[required] : 8443

Principal URL
The Principal URL for the CalDAV account
[optional]

Account Username
The CalDAV username
[optional]

Account Password
The CalDAV password
[optional]

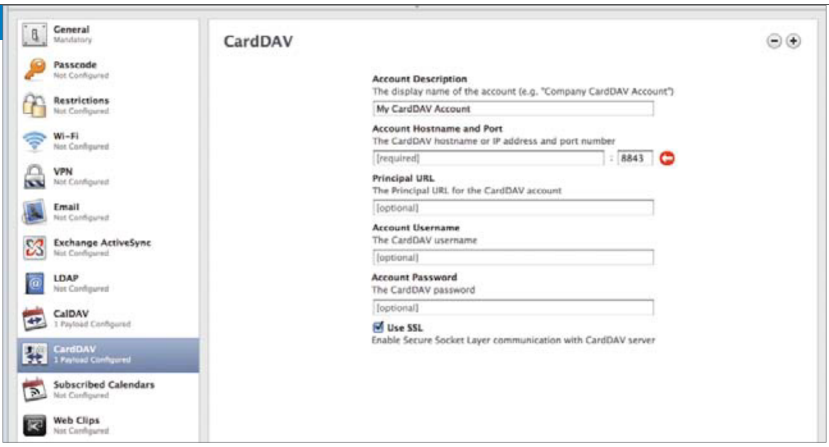
✓ **Use SSL**
Enable Secure Socket Layer communication with CalDAV server

FIGURE 4.11 CalDAV settings

CalDAV is an open calendaring standard used by many companies, Apple and Google among them, to set up calendar servers. Unsurprisingly, iOS supports CalDAV. (A little background: CalDAV combines .ics calendaring files as the data format and WebDAV as the transport mechanism, hence: CalDAV.) The setup is simple with the only oddball bit being the Principal URL (**Figure 4.11**). That's the URL that contains the path to the CalDAV store on the server. This path can vary depending on the server and its implementation. For example, the Principal URL might look like */caldav/users/domain.com/john/*.

As with Email and other "user input required" settings, leave the Account Username and Account Password fields blank to permit the user to enter that information. And always use SSL.

GETTING IN TOUCH WITH **CARDDAV**



CardDAV

Account Description
The display name of the account (e.g. "Company CardDAV Account")
My CardDAV Account

Account Hostname and Port
The CardDAV hostname or IP address and port number
[required] : 8843

Principal URL
The Principal URL for the CardDAV account
[optional]

Account Username
The CardDAV username
[optional]

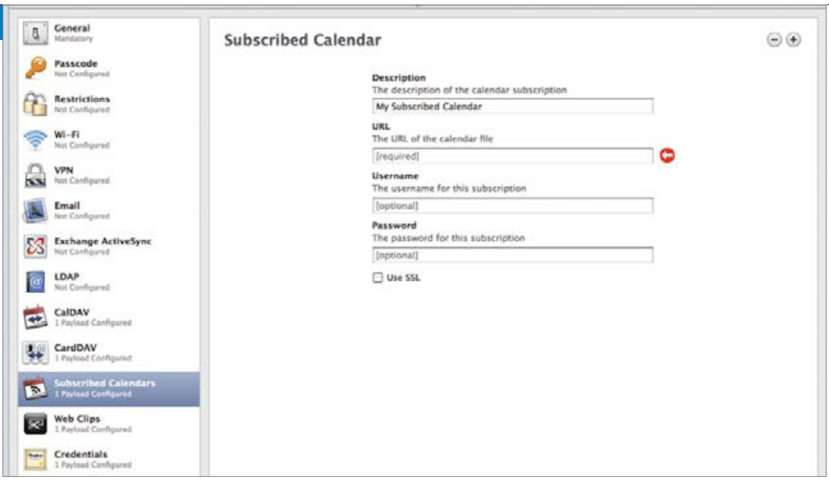
Account Password
The CardDAV password
[optional]

☑ **Use SSL**
Enable Secure Socket Layer communication with CardDAV server

FIGURE 4.12 CardDAV settings

As you saw earlier, LDAP, while usable for email addresses and other contact info, is not always the best choice for many reasons, including that it's kind of a pain to set up. So, a few years back, Apple and several other entities took the CalDAV concept and applied it to contacts. What they came up with was using the vCard format as the data store and, again, WebDAV as the transport mechanism. The result? CardDAV. The setup is identical to CalDAV, including my standard warning to use SSL (**Figure 4.12**).

KEEPING UP WITH **SUBSCRIBED CALENDARS**



General
Mandatory

Passcode
Not Configured

Restrictions
Not Configured

Wi-Fi
Not Configured

VPN
Not Configured

Email
Not Configured

Exchange ActiveSync
Not Configured

LDAP
Not Configured

CalDAV
1 Payload Configured

CardDAV
1 Payload Configured

Subscribed Calendars
1 Payload Configured

Web Clips
1 Payload Configured

Credentials
1 Payload Configured

Subscribed Calendar

Description
The description of the calendar subscription
My Subscribed Calendar

URL
The URL of the calendar file
[required]

Username
The username for this subscription
[optional]

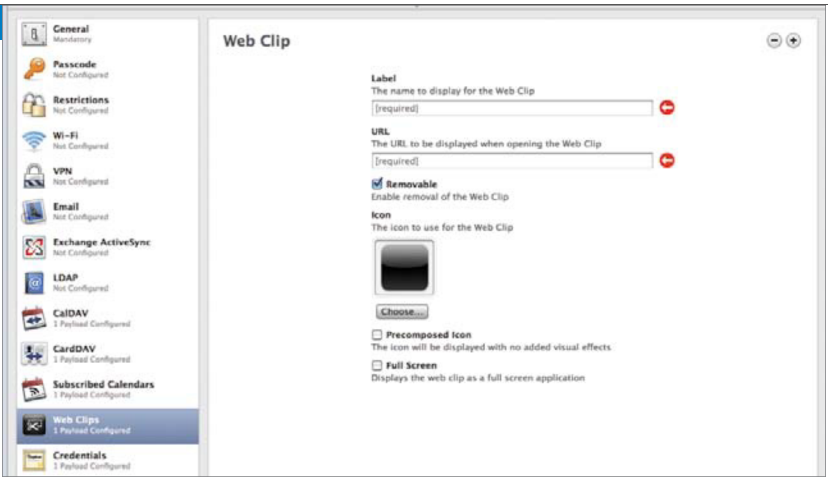
Password
The password for this subscription
[optional]

☐ Use SSL

FIGURE 4.13 Enter username and password in the Subscribed Calendar settings.

These are publicly-accessible, read-only calendars based on the .ics file format that may be used for event lists, team schedules, and so on. Apple has a list of publicly accessible calendars at www.apple.com/downloads/macosx/calendars/. If the calendar in question requires a username and password, those would be entered here (**Figure 4.13**), unless they need to be customized for each user, in which case you'd leave them blank. As always, use SSL if it's an option. (Though, sometimes, you just don't care. I doubt it's a problem if Little Bobby Haxx0r sniffs the updates to my calendar of sunrise/sunset times in Helsinki.)

USING **WEB CLIPS**


General
Mandatory
Passcode
Not Configured
Restrictions
Not Configured
Wi-Fi
Not Configured
VPN
Not Configured
Email
Not Configured
Exchange ActiveSync
Not Configured
LDAP
Not Configured
CalDAV
1 Payload Configured
CardDAV
1 Payload Configured
Subscribed Calendars
1 Payload Configured
Web Clips
1 Payload Configured
Credentials
1 Payload Configured

Web Clip

Label
The name to display for the Web Clip
[required]

URL
The URL to be displayed when opening the Web Clip
[required]

[✓] Removable
Enable removal of the Web Clip

Icon
The icon to use for the Web Clip

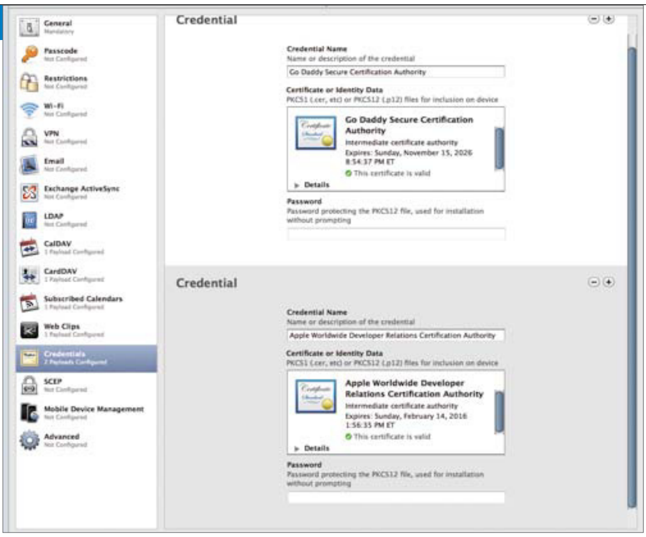
[ ] Precomposed Icon
The icon will be displayed with no added visual effects

[ ] Full Screen
Displays the web clip as a full screen application

FIGURE 4.14 Create web links for the iOS device.

At first glance, these settings can seem kind of useless. "What? Now we don't want them typing in a browser?" But if you think about it, placing web links on an iOS device can be pretty useful for many companies. For example, do you have an internal Wiki or other informational web page? You can set those up here, and make them available on an iOS device without requiring the user to trawl through bookmarks. Do you have custom internal web applications for your users? Enter the info here (**Figure 4.14**), add a pretty icon, and select the "Full Screen" checkbox. When someone taps that link, it looks and behaves like an app, not a web page. Smaller companies may not have many uses for this feature, but for larger companies, these links can be quite handy.

SETTING **CREDENTIALS**



Credential

Credential Name
Name or description of the credential
Go Daddy Secure Certification Authority

Certificate or Identity Data
PKCS1 (.cer, etc) or PKCS12 (.p12) files for inclusion on device

Go Daddy Secure Certification Authority
Intermediate certificate authority
Expires: Sunday, November 15, 2026 8:54:37 PM ET
This certificate is valid

Password
Password protecting the PKCS12 file, used for installation without prompting

Credential

Credential Name
Name or description of the credential
Apple Worldwide Developer Relations Certification Authority

Certificate or Identity Data
PKCS1 (.cer, etc) or PKCS12 (.p12) files for inclusion on device

Apple Worldwide Developer Relations Certification Authority
Intermediate certificate authority
Expires: Sunday, February 14, 2016 1:56:35 PM ET
This certificate is valid

Password
Password protecting the PKCS12 file, used for installation without prompting

FIGURE 4.15 Credentials settings

Here and there I've talked about certificates and SSL. Well, if you have your own web or email servers and you're using SSL, you're going to have your own certificates for them. Or maybe you're using certs for Exchange or your VPN. If you want to install them on the iOS devices that need them, here's where you do it (**Figure 4.15**).

And it's pretty simple. When you click the Add (+) button in the main Credentials pane, you'll see a dialog that asks you to locate the cert file you want to use. Navigate to the file, click Open, and if it's a valid cert file, you're ready to go. As with Email, CalDAV, Exchange ActiveSync, and other settings, you can create multiple entries here. When you install this profile, the certificates are installed on the device, and ready for use.

ABOUT **SCEP**

Simple Certificate Enrollment Protocol (SCEP) can securely add certificates to a device over the air (OTA). In this group of settings, you enter the information needed to use your device with a SCEP server. I'm not going to go into this in any detail here because several later chapters are devoted to SCEP and will provide all the detail you need.

USING **MOBILE DEVICE** MANAGEMENT

Mobile Device Management (MDM) allows you to do everything you've done here in the iPCU, but wirelessly, securely, and with a lot less work on the part of the user. MDM works with Apple's Push Notification Server (APNS) to wirelessly push configurations and configuration changes to iOS devices. As you'll see later in the book, you can do some really cool stuff with MDM. Because Mobile Device Management will be explored in detail later, you can leave these settings alone for now; although if you examine them, you'll begin to see what you can do with MDM, and how useful it can be.

MANAGING **ADVANCED** SETTINGS

Remember back in the "Setting a Passcode" section that I said some things that fall into the "If I have to ask, the answer is probably No" category? The Advanced settings in the iPCU are the perfect example. Here you change your devices' cellular connection settings. Apple says it in the iPCU settings themselves, and I'll repeat it here: *These settings should only be managed by trained professionals.* If you mess these up, your users won't have an iPhone or an iPad with 3G features any more. They have an iPod Touch. The iPod Touch is a neat bit of kit, but not if you need to make a call or use the cellular network to get work done. Unless you have a clear need to change these settings and know *exactly* what you are doing and *why* you are doing it, stay away from these settings like they were a swarm of rabid wolverines.

WRAPPING **UP**

Woohoo! You made it through all the configuration profile settings! Congratulate yourself, you now have a solid bit of knowledge to help you manage and set up your iOS devices. Next, we'll dive a little deeper and look at the structure of the configuration files you'll be creating when you want to install these settings on devices without using a USB cable.

5

UNDERSTANDING **CONFIGURATION PROFILE** STRUCTURE

So now that you've had a detailed look at setting up configuration profiles using the iPhone Configuration Utility (iPCU), let's examine the structure of a profile. This isn't just as an abstract "I know more about profiles than everyone else" exercise. There's actual value in knowing what's going on inside those files.

For example, if you want to customize a profile when you don't have the iPCU handy, knowing how the file is structured and how its data is stored allows you to modify it using almost any text editor. Or, you could automate updates to the configuration utility via various scripting implementations.

If you know the file structure and how the data is used and included, you have more options for configuring your iOS devices.

STARTING WITH THE BASICS

When you open a file with the configuration profile extension .mobileconfig in a text editor, you'll see that it's yet another plist-style XML file. As such, the root structure of the file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.
→apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>PayloadContent</key>
    <array>


    </array>
    <key>PayloadDescription</key>
    <string>This is a profile used for my book figures.</string>
    <key>PayloadDisplayName</key>
    <string>Book Profile</string>
    <key>PayloadIdentifier</key>
    <string>com.bynkii.bookprofile</string>
    <key>PayloadOrganization</key>
    <string>Home</string>
    <key>PayloadRemovalDisallowed</key>
    <true/>
    <key>PayloadType</key>
```

NOTE: The examples in this chapter use an unsigned configuration profile. If you're going to create the configuration profile outside of the iPCU, you'll have to sign it separately. We'll discuss how to do that in this chapter. It's pretty simple.

```
    <string>Configuration</string>
    <key>PayloadUUID</key>
    <string>B9AF31F9-FBB8-4615-8554-317BBD2BE854</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
</dict>
</plist>
```

This structure is in every .mobileconfig file, and contains the main configuration information that you enter in the iPCU "Configuration Profiles" General section. The other sections of the configuration profiles are in `<dict></dict>` blocks in the `<array></array>` block. Because most of the parameters in this part of the profile are identical to the General section of the iPCU, I won't repeat that information here. But I will describe two elements: PayloadUUID and PayloadRemovalDisallowed.

The PayloadUUID number is a universally unique identification (UUID) number, and is a string that is an (obviously) unique identifier for that payload. However, when I say "universally," I am not speaking literally, or even close. The iPCU has no way to know about *every single UUID ever made* and find one that's not been used. So really, it's "universally unique" on your Mac or on your network. Since the PayloadUUID has no real meaning outside of configuration files, as long as the .mobileconfig files you use don't reuse an UUID, it's all good.

So now let's look at PayloadRemovalDisallowed. The concept seems simple enough, in that the profile includes this <*key*>, and it's followed by either <true /> or <false />, which means...wait, in the iPCU, the Security setting includes *three* options for this: Yes, With Authorization, and Never. So, how do we handle three options with a boolean value?

After a bit of trolling through the file (in BBEdit, the most awesome text editor/toolbox *ever*) and browsing Apple's Enterprise Deployment Guide, (which has a *very* detailed item-by-item description of the .mobileconfig file format at

NOTE: You can create your own UUIDs using the uuidgen utility in /usr/bin on your Mac. It's dead simple to use. In a terminal window, enter `/usr/bin/uuidgen`, and you'll get a UUID number that you can use. This utility makes it trivial to generate your own UUID numbers as needed; and if you're going to custom-build your .mobileconfig files, you'll be generating a lot of UUID numbers.

http://manuals.info.apple.com/en_US/Enterprise_Deployment_Guide.pdf), you'll find the answer. Although both "Never" and "With Authorization" have Payload-RemovalDisallowed set to true, "With Authorization" adds a `<dict>` block that contains the information you need to set up the authorization:

```
<dict>
    <key>PayloadDescription</key>
    <string>Configures Configuration Profile security</string>
    <key>PayloadDisplayName</key>
    <string>Profile Security</string>
    <key>PayloadIdentifier</key>
    <string>com.bynkii.bookprofile.ProfileSecurity</string>
    <key>PayloadOrganization</key>
    <string>Home</string>
    <key>PayloadType</key>
    <string>com.apple.profileRemovalPassword</string>
    <key>PayloadUUID</key>
    <string>6A03F3E7-241C-4F6D-AC47-17B05425CF90</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
    <key>RemovalPassword</key>
    <string>testtest</string>
</dict>
```

If you're thinking "Gee, this looks a lot like that payload block I saw earlier," you're right. It does, except that this block configures the profile security, as shown in the PayloadDescription key. (As you'll see, all payloads look much the same.) Looking through this payload block, you'll find the data is well-named (like most plist-style files), so you can easily dope out what each key does. However, do look at the RemovalPassword key. Notice that the password is not encrypted or even obfuscated. Remember that when you're handing out these profile files.

EDITING **INDIVIDUAL PAYLOAD** SECTIONS

Each of the payload sections is contained in a `<dict></dict>` block, and regardless of what a payload configures, all payload sections share some keys:

```
<key>PayloadDescription</key>
<string>Configures email account.</string>
<key>PayloadDisplayName</key>
<string>IMAP Account (Company Account)</string>
<key>PayloadIdentifier</key>
<string>com.bynkii.bookprofile.email</string>
<key>PayloadOrganization</key>
<string>Home</string>
<key>PayloadType</key>
<string>com.apple.mail.managed</string>
<key>PayloadUUID</key>
<string>76D2D26A-E5AB-4CC9-AC44-42E737314A1E</string>
<key>PayloadVersion</key>
<integer>1</integer>
```

The noteworthy keys here are the PayloadType, which identifies the kind of payload (email in this case); the ever-present PayloadUUID; and the version of this particular payload.

When you get past the common keys, each payload has unique information. Here's an entire email section set up for IMAP (of course), with both the incoming and outgoing servers using SSL, HTTP MD5 Digest Authentication, and authenticated SMTP (a.k.a. SMTP AUTH):

```
<dict>
    <key>EmailAccountDescription</key>
    <string>bynkii.com</string>
    <key>EmailAccountName</key>
    <string>John Welch</string>
```

```
<key>EmailAccountType</key>
<string>EmailTypeIMAP</string>
<key>EmailAddress</key>
<string>jwelch@bynkii.com</string>
<key>IncomingMailServerAuthentication</key>
<string>EmailAuthHTTPMD5</string>
<key>IncomingMailServerHostName</key>
<string>mail.bynkii.com</string>
<key>IncomingMailServerPortNumber</key>
<integer>993</integer>
<key>IncomingMailServerUseSSL</key>
<true/>
<key>IncomingMailServerUsername</key>
<string>jwelch</string>
<key>IncomingPassword</key>
<string>testtest</string>
<key>OutgoingMailServerAuthentication</key>
<string>EmailAuthHTTPMD5</string>
<key>OutgoingMailServerHostName</key>
<string>mail.bynkii.com</string>
<key>OutgoingMailServerPortNumber</key>
<integer>465</integer>
<key>OutgoingMailServerUseSSL</key>
<true/>
<key>OutgoingMailServerUsername</key>
<string>jwelch</string>
<key>OutgoingPasswordSameAsIncomingPassword</key>
<true/>
```

```
    <key>PayloadDescription</key>
    <string>Configures email account.</string>
    <key>PayloadDisplayName</key>
    <string>IMAP Account (Company Account)</string>
    <key>PayloadIdentifier</key>
    <string>com.bynkii.bookprofile.email</string>
    <key>PayloadOrganization</key>
    <string>Home</string>
    <key>PayloadType</key>
    <string>com.apple.mail.managed</string>
    <key>PayloadUUID</key>
    <string>76D2D26A-E5AB-4CC9-AC44-42E737314A1E</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
</dict>
```

It's long and dry, but pretty self-explanatory. However, *do* pay attention to the password in the IncomingPassword key. Plain text is easy to program, but it's easy to read too, even by people you don't want reading it. Keep this in mind when you're focusing on configuration file security once you've created the file.

NOTE: Apple does allow you to encrypt these files for security. However, encryption also attaches each profile to a specific device, which does complicate things, because encryption requires that each device have its own profile. But this is not ridiculously onerous.

For the most part, the Enterprise Deployment Guide, and specifically Appendix B, provides every bit of information you'll need to build your own .mobileconfig profile with one annoying exception: configuring credentials. That information isn't in the guide. The workaround is obvious: Create the initial .mobileconfig file in the iPCU, and then copy the credential data from that profile and insert it in your custom profile(s). Still, leaving that information out of the guide is kind of exasperating.

Luckily, thanks to a few very smart people on Twitter, I found the actual format of that payload. It's an NSData *blob*, or just a big binary bit of data created by NSData functions in Cocoa. Why would Apple use a custom solution like an NSData blob instead of just embedding the cert data into the payload? I've no idea. You already can add certs to, for example, web servers by copying the text version of cert data into a new file on the web server; it's not like you gain anything in terms of functionality or even a reduction in file size from this. However, that does mean you have to do a bit more work to create the .mobileconfig file if you want to bypass the iPCU.

You could also use the iPCU to create the credentials payload, and then build the rest as necessary. However you do generate the credentials payload, you'll want to pay attention to the UUID of the cert payload because other payloads reference that cert, such as the Wi-Fi and VPN payloads.

NOTE: **Even if you aren't a Cocoa programmer and know nothing about Objective-C, you can still use NSData. You have a number of ways to use the Cocoa APIs with other languages such as Python, Ruby, or my personal favorite, AppleScriptObjC. You don't have to learn Objective-C and Cocoa programming just to create a .mobileconfig file.**

WHY DO I CARE?

When you can build and modify your own .mobileconfig files, you gain flexibility and features that you don't get when using the iPCU. For example, I previously talked about leaving the user ID fields for email blank to avoid manually changing that for every person using that profile. When you get away from iPCU and apply your knowledge of profile structure, we can programmatically automate the creation of profiles and eliminate the issue entirely (and the need to handcraft every profile).

For example, let's assume that you're tracking users in your company and their email addresses. When you are setting up an account for a new iOS device user, as part of that process, you could use a script that automatically adds her email account info to a shell .mobileconfig file that contains only the credentials and basic root configuration. Grab her user name, server name, and email address, from your tracking database and use it to build the information for the email `<dict></dict>` block.

You could then put that file on a secure web site and direct the user to it. She logs in with her iOS device and installs the profile. All she has to enter is her password, which you'd want her to enter manually anyway. If you're really good with web scripting, you could even set it up so that the server creates the profile when the user logs into the site.

The idea here is that by knowing how and why the .mobileconfig file is structured, you can create custom processes to make iOS device configuration easier. You'll learn more about this in Chapter 6.

SIGNING AND **ENCRYPTING** PROFILES

Okay, so you know how you can create profiles, but how do you sign and encrypt them outside of the iPCU? Signing is pretty simple, encrypting not so much. To sign a profile, you need:

- The .mobileconfig file
- The server certificate you're going to sign with, for example, server.crt
- The private key for that cert, such as server.key
- The cert for the Certificate Authority (CA) that issued your server cert, such as cert-chain.crt
- A name for the signed mobileconfig profile, for example, signed.mobileconfig

If your certs and keys are valid, the process is pretty simple. Just run:

```
openssl smime -sign -in company.mobileconfig -out signed.
→mobileconfig -signer server.crt -inkey server.key -certfile
→cert-chain.crt -outform der -nodetach
```

and you'll have a signed mobileconfig file.

The problem with encryption is that you need the device information because profile encryption needs the device's key to do its job. To do this "easily," you'll want to use a protocol such as SCEP.

However, when you aren't including passwords in your profiles, the need for encryption is reduced somewhat. Furthermore, if the channel you use to distribute the files is secure, the need for encryption drops even further. (If you do need to encrypt your profiles, see Chapters 10–12, where we'll look at SCEP and over-the-air enrollment.)

WRAPPING **UP**

Between understanding .mobileconfig profile structure and reading the Apple Enterprise Deployment Guide (and this chapter, of course), you now have a much better grasp of how to build your own configuration profiles and apply them in your own environment.

The next chapter looks at creating configuration profiles using AppleScript. Even if you don't normally use AppleScript, the methods we'll look at for creating profiles should be of use regardless of which language you prefer to use.

6

SCRIPTING THE iPHONE CONFIGURATION UTILITY

This chapter looks at automating the creation of config profiles via scripting. You'll learn about scripting the iPhone Configuration Utility (iPCU) application itself, and how to bypass the iPCU entirely.

True, the iPCU isn't hard to use. But if you want to create highly customized profiles, change profiles without manually doing so in the iPCU, or integrate profile creation and maintenance as a part of other workflows, you'll want to use scripting.

This chapter is going to revolve around AppleScript because I know it really well, it does the job, and the syntax is somewhat self-documenting. However, you could accomplish the same goal in Python, Ruby, Perl, shell, C#, or almost any other language.

LEARNING **APPLESCRIPT** BASICS

THE APPLESCRIPT LANGUAGE

AppleScript is a vaguely English-like programming/scripting language that allows you to manipulate applications to do things much faster than you could do them manually, and in exactly the same every time. I say "vaguely English-like," because while AppleScript is *almost* English, it tends to not be English in amusing ways. For example, to display the URL of the current tab in Safari, AppleScript says:

```
tell application "Safari"
    set theURL to URL of current tab of window 1
end tell
```

That's somewhat in English, but it's probably *not* how you'd ask another human being for that information. So it's "vaguely" English-like. AppleScript is also a dynamic language. Because it manipulates applications, it relies on each application to implement AppleScript within itself. As each application does different things—and there are no hard rules to much of AppleScript—you get some interesting differences. For example, to do the *exact* same thing in Google Chrome that we did in Safari, the AppleScript is:

```
tell application "Google Chrome"
    set theURL to URL of active tab of window 1
end tell
```

Why is it "active" instead of "current" or vice-versa? You'll have to ask the developers of each application. Let's just say that these application-specific "dialects" make AppleScript "interesting."

NOTE: **For more information about AppleScript, see *Apple Training Series: AppleScript 1-2-3* by Sal Soghoian and Bill Cheeseman, published by Peachpit Press. The Internet also has many good resources such as Apple's AppleScript Users' email list (http://lists.apple.com/mailman/listinfo/applescript-users), and MacScripter (www.macscripter.net/).**

THE DICTIONARY

In AppleScript, you use the *dictionary* to see what scripting functionality an application supports. *Everything that can be scripted has a dictionary*. Even the base Mac OS X has one, called, amusingly, "Standard Additions." (I'll ignore the grammatical silliness of that. But it is some mighty silliness.) The Finder, Microsoft Word, and all AppleScript-able applications have a dictionary, including the iPCU. Some dictionaries are fantastic. (For all the ragging they get, the core Adobe CS suite applications have *phenomenal* dictionaries.) Others are less so. (We'll not name names here. No sense in further embarrassing the wicked.) The iPCU's dictionary isn't the best I've ever seen, but it provides the necessary capabilities.

SCRIPTING THE IPHONE CONFIGURATION UTILITY

I'm not going to go through every possible way to script the iPCU because this book would become infinitely long (and I'm not getting paid by the word. Alas!) However, let's look at a few settings to give you a feel for what's possible, and also to show you how to make a script flexible and fast, so you can customize .mobileconfig files with very little effort.

One thing to be clear about: you can only script configuration file setup with AppleScript. You can't script application installs, and so on. Now, let's look at a basic setup with email, CalDAV, and some restrictions.

USING **SCRIPT EDITORS**

To create, edit, and test AppleScripts, you obviously need a script editing application. Apple includes one as part of Mac OS X: Script Editor in earlier versions of Mac OS X, AppleScript Editor in the current version, Mac OS X 10.6. For *basic* scripting, AppleScript Editor is decent. However, if you start writing longer scripts, you're going to tire of its limitations, particularly when debugging scripts. If you are going to be scripting regularly (and why wouldn't you?), run—do not walk—to Late Night Software, and buy Script Debugger (www.latenightsw.com/sd4/index.html). It is a phenomenal tool that gives you everything you need to be a better scripter. It's not free ($199 US at the time of this writing), but it is worth every penny. I cannot imagine writing scripts without it.

The first thing you need to do in the script is make sure that it is targeting the iPCU; otherwise, you'll have some serious issues. To do that, you use a *tell* block:

```
tell application "iPhone Configuration Utility"

end tell
```

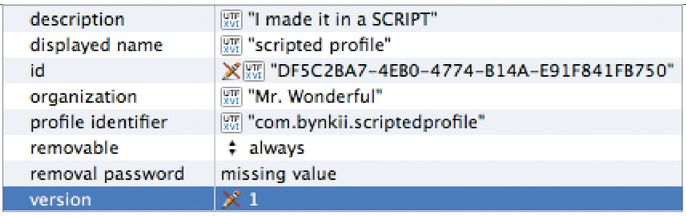
You put all your code between those two lines, and your script will work much better. Remember that some basic items—such as the name—*have* to be in every profile, even though they are of no functional use. So, when you create the initial profile, you set those properties at the same time:

```
tell application "iPhone Configuration Utility"

    set theProfile to make new configuration profile with properties
    →{displayed name:"scripted profile", profile identifier:
    →"com.bynkii.scriptedprofile", organization:"Mr. Wonderful",
    →account description:"I made it in a SCRIPT"}

end tell
```

Part of this script is pretty obvious. It's telling the iPCU to create a new profile called "theProfile." It has a list of properties such as the displayed name, the identifier, the organization, and the account description. AppleScript *properties* are created as an AppleScript *record*—that is, a comma-delimited list of elements that have an identifier and a value. So, in this case, an identifier is `displayed name`, and its value is `"scripted profile"`. Since "scripted profile" is text, you enclose it in quotes to identify it as such. A colon separates the identifier and its value, and the whole thing is enclosed in curly braces. So, a generic record of text values would be:

```
{identifer1:"one",identifier2:"two",identifier3:"three"}
```

and so on. All AppleScript properties are records.



description	"I made it in a SCRIPT"
displayed name	"scripted profile"
id	"DF5C2BA7-4EB0-4774-B14A-E91F841FB750"
organization	"Mr. Wonderful"
profile identifier	"com.bynkii.scriptedprofile"
removable	always
removal password	missing value
version	1

FIGURE 6.1 iPCU creates some values automatically.

However, in the properties of this newly-created profile (**Figure 6.1**), you'll see some things you didn't create, like the id, removable, and the removal password.

Those values are needed, but in the case of the ID, you *can't* create it, at least not via the iPCU. That's okay because the iPCU does it for you. In the case of the removal and the removal password values, you didn't specify them, so the iPCU set them to default values, "always" and "missing value" (a.k.a. "nothing"), respectively. That's one of the advantages of scripting an application: the script can do a lot of the scutwork so you don't have to.

Now that you have a base config, you can add some payloads; but first, a word about *elements*. An AppleScript element is a part of a class that can have its own properties. In this case, the main class is "configuration profile." The configuration profile has its own properties, such as the id, organization, and so on. The payloads are classes that are also elements.

Why not make them properties, since they're a part of the configuration profile? Because then you'd have a properties record that was insanely long and really hard to change or adjust. As you'll soon see, by implementing the payloads as elements of a configuration profile, you can make things more modular, and easier to deal with.

One neat thing about AppleScript and the "tell" concept is that it's not just for applications. You can use it for classes, too. So now that we created a configuration profile, we're going to "tell" it about the payloads it's getting. The first payload will be a restrictions payload. To add that, add the following code below the initial line that creates the configuration profile:

```
tell theProfile

    make new restrictions payload with properties {allow adding
    →game center friends:false, allow multiplayer gaming:false,
    →explicit content allowed:false, YouTube allowed:false}

end tell
```

This bit's pretty clear, right? The only real difference with these properties is that the values are all *boolean* values, and are either true or false. Since they're not text values, they don't need the quotes. Luckily, we don't care about the id because the iPCU takes care of this for you when you save the profile to a .mobileconfig file.

Next, create an email profile by adding in the appropriate "make new email payload" section:

```
make new email payload with properties {account description:
→"scripted email account", account name:"John C. Welch",
→account protocol:IMAP, email address:"john@bynkii.com", incoming
→server hostname:"imap.bynkii.com", incoming server port:143,
→incoming server username:"john", incoming server uses password
→authentication:true, incoming server uses SSL:false, outgoing
→server hostname:"smtp.bynkii.com", outgoing server port:587,
→outgoing server username:"john", outgoing server uses password
→authentication:true, outgoing server uses SSL:false, use incoming
→password when sending mail:true}
```

No need to memorize this stuff, other than specific values such as the incoming server username's value. It's all in the iPCU dictionary. Now, for your CalDAV payload:

```
make new CalDAV payload with properties {account description:
→"scripted CalDAV account", hostname:"calendar.bynkii.com",
→port:80, principal URL:"http://calendar.bynkii.com/caldav",
→use SSL:false, user name:"john"}
```

Finally, you'll need to tell the iPCU where to save the profile and with what filename:

```
export theProfile to "/Users/jwelch/Desktop/test.mobileconfig"
```

So, when we put the whole script together, we get:

```
tell application "iPhone Configuration Utility"
    set theProfile to make new configuration profile with properties
    →{displayed name:"scripted profile", profile identifier:
    →"com.bynkii.scriptedprofile", organization:"Mr. Wonderful",
    →account description:"I made it in a SCRIPT"}
    --set theEmailUsername to "john"
    tell theProfile
        make new restrictions payload with properties {allow adding
        →game center friends:false, allow multiplayer gaming:false,
        →explicit content allowed:false, YouTube allowed:false}
        make new email payload with properties {account
        →description:"scripted email account", account name:"John
        →C. Welch", account protocol:IMAP, email address:"john@
        →bynkii.com", incoming server hostname:"imap.bynkii.com",
        →incoming server port:143, incoming server username:"john",
        →incoming server uses password authentication:true,
        →incoming server uses SSL:false, outgoing server
        →hostname:"smtp.bynkii.com", outgoing server port:587,
        →outgoing server username:"john", outgoing server
        →uses password authentication:true, outgoing server uses
        →SSL:false, use incoming password when sending mail:true}
        make new CalDAV payload with properties {account
        →description:"scripted CalDAV account", hostname:"calendar.
        →bynkii.com", port:80, principal URL:"http://calendar.
        →bynkii.com/caldav", use SSL:false, user name:"john"}
    end tell
    export theProfile to "/Users/jwelch/Desktop/test.mobileconfig"
end tell
```

Pretty nice, and the results are a properly configured .mobileconfig file. That's all well and good; but now, when you need to generate a new profile, you have to open the script and change the name. So what have you saved in terms of time and effort? Not much.

However, don’t despair, for you can fix this with ease. What you need to do is add a way to change just the user names without manually editing the script. To do that, you’ll add some code so that the script *asks* for the new name and then sets it. For simplicity’s sake, I’m going to assume all user names are the same for every account that the profile is configuring. To get this information, you’re going to use a command from the aforementioned “Standard Additions”: *display dialog*. Display dialog is a way to ask someone to enter a bit of text, and then use that text elsewhere. Display dialog returns a *record*, called the *dialog reply*, that includes three items:

- button returned: the name of the button clicked in the dialog
- gave up: the dialog time out (You can set a time out value if you wish. We won’t.)
- text returned: the text the human types. This is what we care about.

So, here’s the code:

```
set theUserName to text returned of (display dialog "please enter
→the user name" default answer "Bob")
```

Normally, you would set the display dialog line to something that would become the dialog reply, and then you’d pull the text returned from that. To save space and typing, we’ll just do it all on one line. We tell Standard Additions to set theUserName to the text returned from the display dialog command (the only “applications” you don’t need a tell block for are scripting additions, such as Standard Additions). The display dialog command is in parentheses because we want it to execute first, followed by the rest of the line.

We’ll put this line *outside* the iPCU tell block for two reasons: First, the line has nothing to do with the iPCU. This is just good AppleScript practice. Second, it’s safer, although technically speaking, we don’t have to because Scripting Additions is omnipresent in AppleScript. Still, remember the little Chrome/Safari example earlier in this chapter? Well, *sometimes* application vendors are not as careful with names as they should be. Sticking a command from one dictionary into another is a way to get some really odd errors that will make you crazy. In the interest of avoiding the crazy-making, we shan’t be so silly.

You can then change the individual payloads. Wherever an option for a user name was present, change that value from the string it had, ("john" in this example) to theUsername. So you go from this:

```
username:"john"
```

to this:

```
username:theUserName
```

Anytime this script is run, it asks for a user name, and that name becomes the user name for every payload that needs one. If you need multiple user names, you'd use multiple display dialog lines and save the results to different variable names.

So, the script now looks like this:

```
set theUserName to text returned of (display dialog "please enter
→the user name" default answer "Bob")

tell application "iPhone Configuration Utility"

    set theProfile to make new configuration profile with properties
    →{displayed name:"scripted profile", profile identifier:"com.
    →bynkii.scriptedprofile", organization:"Mr. Wonderful", account
    →description:"I made it in a SCRIPT"}

    tell theProfile

        make new restrictions payload with properties {allow adding
        →game center friends:false, allow multiplayer gaming:false,
        →explicit content allowed:false, YouTube allowed:false}

        make new email payload with properties {account
        →description:"scripted email account", account name:"John
        →C. Welch", account protocol:IMAP, email address:"john@
        →bynkii.com", incoming server hostname:"imap.bynkii.com",
        →incoming server port:143, incoming server username:
        →theUserName, incoming server uses password authentication:
        →true, incoming server uses SSL:false, outgoing server
        →hostname:"smtp.bynkii.com", outgoing server port:587,
        →outgoing server username:theUserName, outgoing server
        →uses password authentication:true, outgoing server uses
        →SSL:false, use incoming password when sending mail:true}
        make new CalDAV payload with properties {account
        →description:"scripted CalDAV account", hostname:"calendar.
        →bynkii.com", port:80, principal URL:"http://calendar.
        →bynkii.com/caldav", use SSL:false, user name:theUserName}
    end tell
    export theProfile to "/Users/jwelch/Desktop/test.mobileconfig"
end tell
```

For one at a time needs, this script is great. But if you wanted to be really slick, you could create a modified version that could, say, read a *bunch* of names from a file and then create multiple config profile files in one fell swoop. That's just the ticket when you have a lot of people who need devices set up. For this task, we'll make two assumptions to save space: all user names are the same, and the name in your email address is your user name.

First, add some new lines to the beginning of the script:

```
set theSourceFile to choose file
set theFileReference to open for access theSourceFile without write
→permission
set theNames to read theFileReference
close access theFileReference
set theNameList to every paragraph of theNames
```

These lines do several things. First, they ask you to pick a file that you want to use as a source for user names and email addresses. In this case, I created a text file with one name per line. When you choose this file, the file and path to that file are put into a variable called "theSourceFile".

Next, you want to open theSourceFile to read the data. That's what the "open for access" line does. Because you don't need or want to change any data in the file, you disallow write access. This line creates a file reference number that is put into "theFileReference".

Then, the script reads the data out of the file that theFileReference points to, and shoves it into theNames. This is a bunch of text with the names and the line-ending characters as its content.

To be neat, close access to theFileReference. You don't need it anymore, so closing it is the correct thing to do.

Since we want to make that text in theNames into something easier to use, turn it into a list by getting *every* paragraph in theNames, and putting that into theNameList. A *list* is like a record, only it has no identifiers, just values separated by commas. When you ask for *every* anything in AppleScript, the answer is always a list. In this case, you want every paragraph, which is text separated by returns. Doing so creates a list with each name as its own entry.

So, you now have a list of names.

Now, modify the tell block for the iPCU. Just after the `tell application "iPhone Configuration Utility"` line, add some new lines:

```
repeat with x in theNameList
    set theName to contents of x
    set theUserName to theName
    set theEmailAddress to theName & "@bynkii.com"
    set theFileName to theName & ".mobileconfig"
    set theExportPath to "/Users/jwelch/Desktop/mobileconfigs/"
    →& theFileName
```

A *repeat loop* is just that. It goes round and round doing things over and over until you tell it to stop. The `repeat with x in theNameList` line tells AppleScript that you want to go through every item in theNameList and every time you go through, shove the next item in the list into x. You then set theName to the *contents* of x because what is in x can be inconsistent. Sometimes it's the contents of the list item, or sometimes it's something not so useful such as `item 1 of theNameList`. By using the *contents* of x, you always know that you're getting the name, and not a description of the place in the list where that name is.

Then use theName to create a few variables:

- theUserName (which you've seen before)
- theEmailAddress, created by *concatenating* theName and "@bynkii.com" into one text string (& is the concatenation operator in AppleScript)
- theFileName, by concatenating theName and ".mobileconfig"
- The path for the exported file, by concatenating the path to the destination folder, /Users/jwelch/Desktop/mobileconfigs/ and theFileName

Next, we go through our payload sections and make some substitutions. Wherever a user name is needed, that value is set to theUserName. Wherever an email address is needed, that value is set to theEmailAddress.

We also change our export line to read:

```
export theProfile to theExportPath
```

so that we create a separate .mobileconfig file for each name in the list, with the name as part of the filename. Finally, between the "export" line, and the "end tell" line, add an "end repeat" line to close the repeat statement correctly. The script now looks like this:

```
set theSourceFile to choose file

set theFileReference to open for access theSourceFile without write
→permission

set theNames to read theFileReference

close access theFileReference

set theNameList to every paragraph of theNames

tell application "iPhone Configuration Utility"

    repeat with x in theNameList

        set theName to contents of x

        set theUserName to theName

        set theEmailAddress to theName & "@bynkii.com"
        set theFileName to theName & ".mobileconfig"

        set theExportPath to "/Users/jwelch/Desktop/mobileconfigs/"
        →& theFileName

        set theProfile to make new configuration profile with
        →properties {displayed name:"scripted profile", profile
        →identifier:"com.bynkii.scriptedprofile", organization:"Mr.
        →Wonderful", account description:"I made it in a SCRIPT"}

        tell theProfile

            make new restrictions payload with properties {allow
            →adding game center friends:false, allow multiplayer
            →gaming:false, explicit content allowed:false, YouTube
            →allowed:false}

            make new email payload with properties {account
            →description:"scripted email account", account
            →name:"John C. Welch", account protocol:IMAP, email
            →address:theEmailAddress, incoming server hostname:
            →"imap.bynkii.com", incoming server port:143, incoming
            →server username:theUserName, incoming server uses
            →password authentication:true, incoming server uses
            →SSL:false, outgoing server hostname:"smtp.bynkii.
            →com", outgoing server port:587, outgoing server
            →username:theUserName, outgoing server uses password
            →authentication:true, outgoing server uses SSL:false,
            →use incoming password when sending mail:true}

            make new CalDAV payload with properties {account
            →description:"scripted CalDAV account", hostname:
            →"calendar.bynkii.com", port:80, principal URL:"http://
            →calendar.bynkii.com/caldav", use SSL:false, user name:
            →theUserName}

        end tell

        export theProfile to theExportPath

    end repeat

end tell
```

You added a total of 12 lines of code, and changed 2 or 3 others, but created a script that can create 1 to 1000 individual config profiles, and the only change in effort is creating a list of names. The only other manual task is pointing the script at that name file.

That is why you write scripts; rather than creating all these config files by hand, you can create a bunch of profiles at the same time with a script, and you create them at machine speed, not human speed. Totally sweet.

WRAPPING **UP**

To be honest, you don't *need* the iPCU to script .mobileconfig files. They're just text files with predictable sections and a predictable layout. You don't even need the iPCU to create the UUID numbers, you can do that using uuidgen.

However, you're going to do a lot more work creating the script without iPCU because now you will have to replicate the *entire* XML file structure by hand; and if that structure changes, who gets to change the script to account for that? You do! You're also not going to gain much of a speed improvement. Even with the iPCU's processing overhead, it's going to create profiles automatically *far* faster than you ever will manually.

But, if you don't want to use AppleScript, or C# on Windows, you're kind of stuck doing it the hard way. If you can do that kind of scripting, however, you probably don't need me to tell you how to do it. Spending some time with the iPCU and analyzing several .mobileconfig files will tell you all you need to know.

Whew, that was a lot. And if you've never seen AppleScript, this chapter felt even deeper. I'll warn you, though. Once you get a taste for automating your work, you'll quickly start automating more of your work. Wait! I'm not warning you, I'm *encouraging* you to do this. Life is too short to spend it doing repetitive scutwork on a computer. Automate, automate, automate!

This page intentionally left blank

PART II

OVER-THE-AIR **SETUP**

7

ADDING **PROFILES** TO **DEVICES**

So you've created profiles in every way possible. You have scripts and XML files, along with configuration profiles that fit every possible iOS device and user need. What could possibly be left? Oh, that's right: actually getting the profile from the iPhone Configuration Utility (iPCU) onto the device. In this chapter, you'll learn about two methods: tethered installation and email.

USING A **TETHERED PROFILE** INSTALLATION

The tethered profile installation method is the simplest. Plug an iOS device into a computer running the iPCU. Wait until the device shows up in the Sidebar. Select the device, click the Configuration Profiles tab, find and select the profile you want to install, and click Install (**Figure 7.1**). When you're done, you'll see the standard Install Profile screen on your device (**Figure 7.2**).

On the device, tap Install to install the profile. You can uninstall the profile on the device itself, or by using the iPCU. To remove the profile from the device, go into Settings, General, and Profiles. Tap the profile you want to remove and then tap Remove (**Figure 7.3**).

To remove the profile using the iPCU, plug the device in to a computer and select it. In the iPCU, click the Configuration Profiles tab, select the installed profile, and then click Remove.

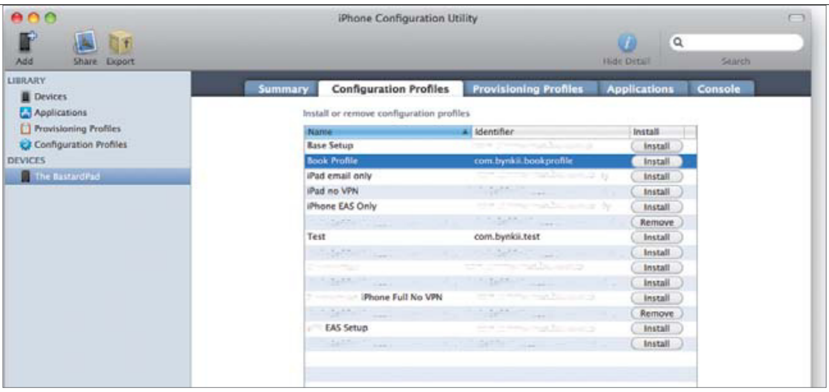
Tethering is simple, fast, and intuitive. It's also secure because the only way to install a profile via tethering is to establish a physical connection between the device and the computer running iPCU.

Tethering is also more flexible than you might think. Usually, the "tethering" suggests some poor IT sap sitting in a room with a stack of iOS devices swapping them in and out while wondering if he should have accepted that management job at the local fast food joint. But with a bit of imagination, you can avoid this scenario (and the burger-flipping thing, too).

First, keep in mind that you don't need a particularly fast computer to install profiles. Can it run Mac OS X v10.6 or even a vaguely new version of Windows? (I'm not kidding about "vaguely"—iPCU supports XP SP3.) If so, it's good enough to run the iPCU.

So, let's say you've issued a big bunch of new iOS devices, and you want to get their users in and out as fast as possible. Here's one option: Set up some cheapie Windows boxes or older Mac Minis as kiosks running only the iPCU. Attach a Dock Connector cable to the kiosk. Put up some nice signage explaining the few steps required to install the company profile. Place these kiosks around the company in appropriate places. Congratulations, you have self-service configuration kiosks!

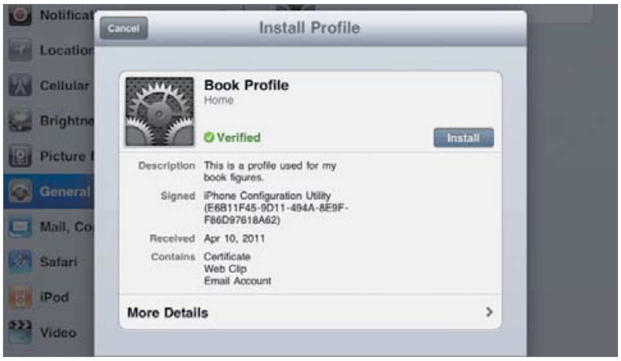
I wouldn't do this on a permanent basis. But if I had very little time to process a shipload of iOS devices, and I had the extra CPUs sitting around, it's not a bad solution. More importantly, it means *you* don't have to install everything yourself.



Install or remove configuration profiles

Name	Identifier	Install
Base Setup	[illegible]	Install
Book Profile	com.bynkii.bookprofile	Install
iPad email only	[illegible]	Install
iPad no VPN	[illegible]	Install
iPhone EAS Only	[illegible]	Install
[illegible]	[illegible]	Remove
Test	com.bynkii.test	Install
[illegible]	[illegible]	Install
[illegible]	[illegible]	Install
[illegible]	[illegible]	Install
iPhone Full No VPN	[illegible]	Install
[illegible]	[illegible]	Remove
EAS Setup	[illegible]	Install
[illegible]	[illegible]	Install

FIGURE 7.1 iPCU with iPad attached, ready to install a profile

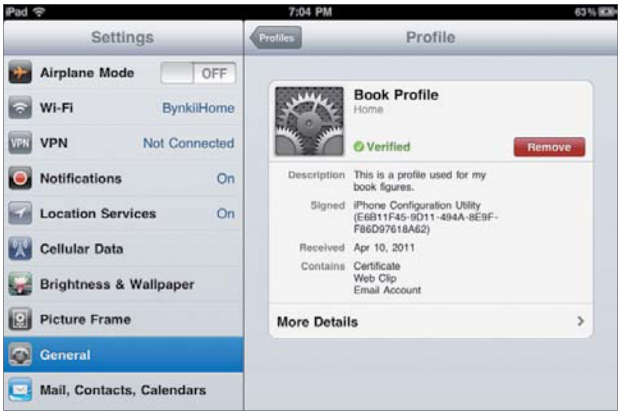


Install Profile

Book Profile
Home
Verified

Description This is a profile used for my book figures.
Signed iPhone Configuration Utility (E6B11F45-9D11-494A-8E9F-F86D97618A62)
Received Apr 10, 2011
Contains Certificate
Web Clip
Email Account

FIGURE 7.2 Install Profile screen on an iPad



Airplane Mode	OFF
Wi-Fi	BynkiiHome
VPN	Not Connected
Notifications	On
Location Services	On

Book Profile
Home
Verified

Description This is a profile used for my book figures.
Signed iPhone Configuration Utility (E6B11F45-9D11-494A-8E9F-F86D97618A62)
Received Apr 10, 2011
Contains Certificate
Web Clip
Email Account

FIGURE 7.3 Installed profile with Remove button on an iPad

INSTALLING WITH **EMAIL**

Email is the other installation method that the iPCU directly supports. It's also pretty simple.

In the iPCU, select the profile you want to send, and click the Share button on the toolbar; or from the File menu, choose Share via Email.

You'll be asked if and how you want to sign the profile. You can choose not to sign the profile, to sign the profile, or to sign and encrypt the profile to a specific device you've attached at least once to the iPCU. Note that if you encrypt to a device, the iPCU will create one profile per device in Mail on Mac or Outlook in Windows (**Figure 7.4**).

If you select multiple devices, however, the iPCU will create just one email containing all of those profiles. I'm not really sure about the logic here. If I'm setting up multiple devices, it seems logical that each device is owned by a different person. With that in mind, I think that iPCU should create multiple email messages, one per profile. Evidently, Apple disagrees.

Oh, and yes, you'll be using Mail or Outlook—at least on Mac OS X because Mail is hardcoded here. Even if you're not using Mail as your default email application, you will be using it to send iPCU profiles.

On the iOS device, you'll receive an email message with the configuration profile as an attachment (**Figure 7.5**). Tap the attachment, and you'll see the standard Install Profile screen on the device, and everything installs as normal.

If you're running Mac OS X Server 10.7, then you don't need the iPhone Configuration Utility. You'll use Profile Manager, but perhaps not in the most obvious of ways. First, open up Server.app, select Profile Manager, and click the link to open the Profile Manager web page (**Figure 7.6**). Log into the Profile Manager page and go to the Devices Section.

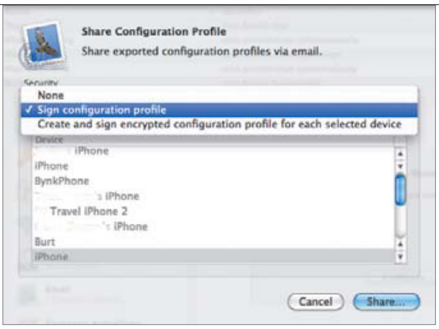
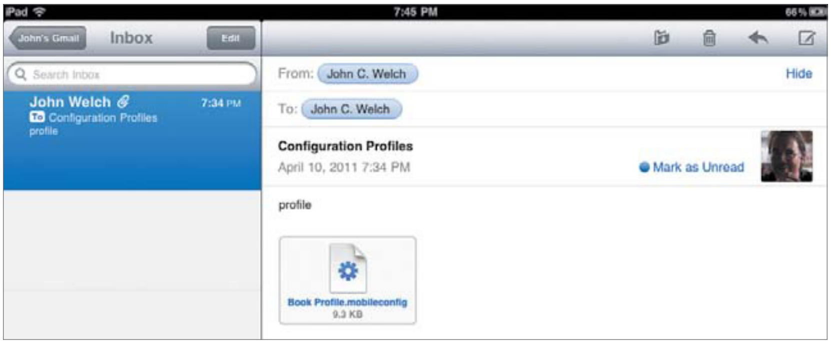

Share Configuration Profile
Share exported configuration profiles via email.
Security
None
✓ Sign configuration profile
Create and sign encrypted configuration profile for each selected device
Device
iPhone
iPhone
BynkPhone
's iPhone
Travel iPhone 2
's iPhone
Burt
iPhone
Cancel
Share...

FIGURE 7.4 The Share Configuration Profile sheet in the iPCU



John Welch 7:34 PM
To Configuration Profiles
profile

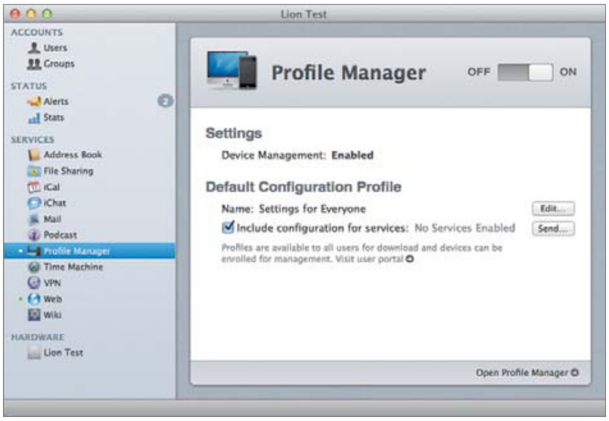
From: John C. Welch
To: John C. Welch

Configuration Profiles
April 10, 2011 7:34 PM

profile

Book Profile.mobileconfig
9.3 KB

FIGURE 7.5 Email message with configuration profile as attachment



Settings
Device Management: Enabled

Default Configuration Profile
Name: Settings for Everyone
✓ Include configuration for services: No Services Enabled
Profiles are available to all users for download and devices can be enrolled for management. Visit user portal

FIGURE 7.6 Profile Manager in Server.app

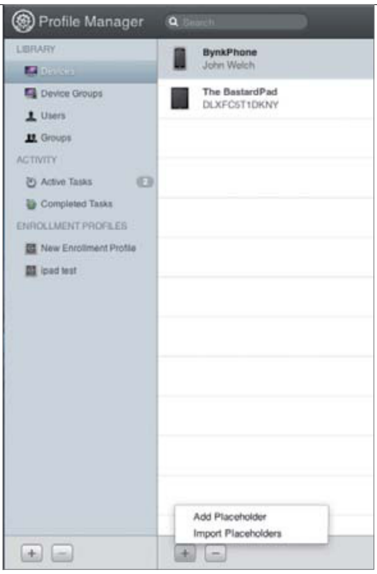


FIGURE 7.7 Adding a placeholder for a device in the Profile Manager web UI

Now, because you want to email the profile to the device, you can't use some of the other options Mac OS X Server 10.7 gives you. You also have to create the profile before you email it. To do that, at the bottom of the column where devices show up, click the + button and select Add Placeholder (**Figure 7.7**). This will let you create a dummy profile for that specific device that you can then set up and download. Because this profile is device-specific, you'll need to provide some identifying information, such as the serial number, UDID, etc. Once that's done, you'll get a dummy profile that you can then configure as needed. When you're finished, click the Download button (**Figure 7.8**) and you'll get a .mobileconfig file you can email to the user. Note that this creates a signed profile in many cases, so editing it directly via some other method may cause problems.

The email distribution method is handy, it uses a ubiquitous distribution mechanism, and it can be somewhat secure. However, when distributing profiles to new devices, it has one rather large air gap: How do you send a profile with email account configuration info via email to a device that doesn't yet have an *email account set up on it?*

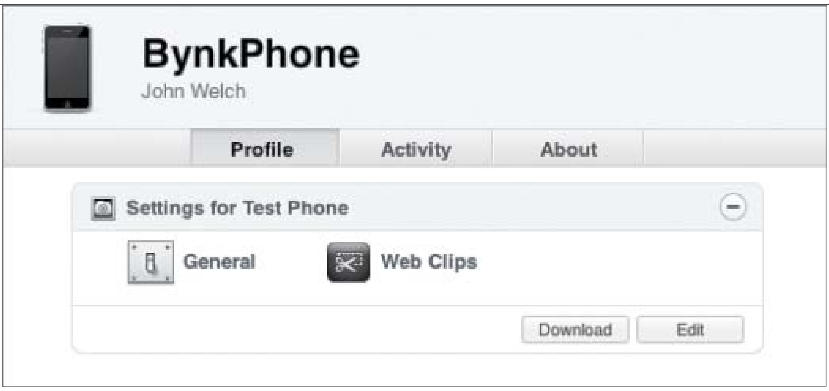


FIGURE 7.8 Download button for a device profile

Webmail is your friend here. On an iPad, this works fairly well because you have a nice big screen. On an iPhone, you may have some unhappy, squinting users—particularly if your webmail provider is overly clever with its "mobile" version. Still, even on small screens, webmail is definitely usable.

I use this distribution method almost exclusively for people who have problems with bad cell phone coverage, or when dealing with a Wi-Fi provider who's just a little too clever with the ports they're blocking. Even if they're blocking IMAP/POP, almost no one blocks HTTP/HTTPS, so the nigh-ubiquitous Gmail can be a lifesaver. (On a lark, I tried, unsuccessfully, to use both Dropbox and iDisk as ways to distribute profiles. Evidently, neither of them is set up for this sort of thing. That's a bit of a shame, because a company Dropbox account could be a neat distribution center.)

WRAPPING **UP**

What about using over-the-air (OTA)? I know, I know. You're not asking about email, you're asking about "real" OTA. That's coming in the next few chapters. In fact, Chapter 8 discusses some quick and dirty (and not terribly complicated) methods for setting up OTA profile distribution.

8

USING SIMPLE OVER-THE-AIR **PROFILE DISTRIBUTION**

You've seen how to use iPhone Configuration Utility (iPCU) to distribute profiles via USB connection and email. But over-the-air, or OTA, is the cool (and best) way to distribute profiles for a couple reasons.

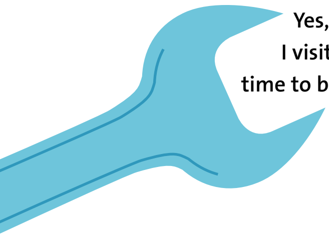
First, you don't have to do all the work yourself. Set up your OTA implementation, and unless you need to change configurations, you're finished. Everything else is done by the user. Second, you can service hundreds or even thousands of users from a single server, and you can allow users to change profiles without coming to you.

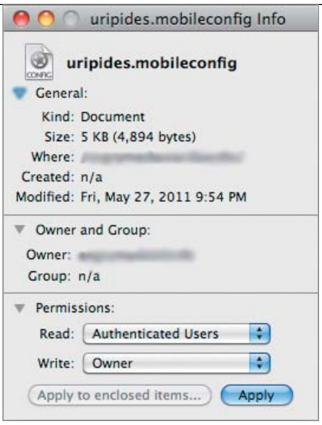
If someone loses his device or has it stolen while traveling, he can get a new device and activate it from wherever he is. Then he can reinstall the profile and be back in business in minutes.

START WITH A **WEB SERVER**

While OTA can become quite complex, you can keep things simple if you choose. In its most basic form, OTA configuration profile distribution requires only a web server. You upload the profile to the web server and provide your users a URL that links to the file. They navigate to that URL on their devices, and the profile installation starts automatically.

NOTE: The OTA benefits for stolen devices are not theory. My iPhone was stolen during Macworld Conference & Expo 2009 while I was on a break from teaching an all-day session. All I did was go to the Apple Store in San Francisco, get a new iPhone 3GS, and by the time I'd walked back to my hotel, all my work settings were reinstalled thanks to my OTA setup. Yes, I knew how to enter in all that stuff manually, but thanks to OTA, I didn't have to. I visited one website, and less than three minutes later, I was set. It took about as much time to brick my old phone as set up the new one. OTA is powerful.




uripides.mobileconfig Info

uripides.mobileconfig

General:

Kind: Document
Size: 5 KB (4,894 bytes)
Where:
Created: n/a
Modified: Fri, May 27, 2011 9:54 PM

Owner and Group:

Owner:
Group: n/a

Permissions:

Read: Authenticated Users
Write: Owner

FIGURE 8.1 Settings permissions

USING AMAZON'S S3 SERVICE

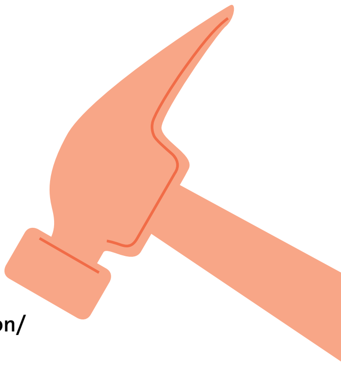
You don't even need to own your web server (although for increased security, you'll want to). Thanks to a reply from @abrahamvegh on Twitter, I realized that you could use Amazon's S3 service to distribute configuration profiles. You wouldn't want to make the profile world-readable, but it definitely works, and it's pretty simple:

1. Upload the .mobileconfig file to your S3 account.
2. Make sure to set the permissions for the file so that authenticated users can read it (**Figure 8.1**).
3. Give those users the S3 URL for the .mobileconfig file.

That's really it. There is one caveat to this process: Amazon only lets you set up other S3 users as "authenticated" users. So while it's neat that you can use S3 for this task—and it really is simple to set up and use—it's probably not the most practical idea for company-wide use. (Although, if I needed a reliable service to distribute profiles with noncritical data, such as the password to a hotel Wi-Fi network, putting the profiles on S3 and setting them to be world-readable would not be the worst solution.)

But now let's look at doing "simple" OTA profile distribution the right way.

TIP: The site Bucket Explorer has great information on what constitutes "authenticated users" in S3 parlance at www.bucketexplorer.com/documentation/amazon-s3--access-control-list-details.html.



SETTING UP THE **OTA WEB SERVER**

Obviously, we have a few goals here:

- Set up the server to be as secure as possible.
- Restrict the server so not just anyone has access to it.
- Make it simple for nontechnical people to use.

Luckily, all those goals are achievable. I'm basing this chapter on my own setup, which is on a Mac OS X Server that is part of an Open Directory network. However, you don't need to use Mac OS X Server or Open Directory. That just happens to be what my network mostly is.

First, I picked a server that I didn't mind making accessible to the outside world through my firewall. That's important, because even though I configured the server in a reasonably secure manner, I don't want to expose important data any more than necessary. In this case, the server was already an FTP server, so it was a good choice to use here. It also had all the requisite firewall rules set up, which was handier still. The fewer holes you have to poke through a firewall, the better.

Next, I created an SSL-enabled website on the server and pointed it at the directory that would contain the .mobileconfig files. SSL is obviously important, as I don't want any part of OTA profile distribution occurring in an unencrypted format, or "in the clear."

For the site itself, I disabled everything I didn't absolutely need in order to distribute configuration profiles. So the server offers no folder listings, no CGIs, no WebDAV, no nothing that isn't needed to download a file to an iOS device. Other than the .mobileconfig files in the folder, the only file found here is an index.html file that says, "These aren't the droids you're looking for." That's mostly there for people who enter the wrong URL for the .mobileconfig file. It's pretty difficult to just "stumble across" this site. So even if an outsider gets into the server, she's not going to find much. And by turning off almost every bit of advanced web server functionality, the chances of that server being hacked go way down.

I also took advantage of the ability of Mac OS X Server to tie website access to Open Directory user accounts. Even if you have the specific URL for the server, you must still have a valid Open Directory account to access it. This does several things, all of them good. I don't have to set up user accounts or passwords. When people need to get to a profile, they already know the user name and password for it because it's the same login they use for all other company business. Because that user name and password are tied to Open Directory, if someone leaves the company, disabling his Open Directory account disables his access to the profile. Because the site is SSL-enabled, everything happens over an encrypted channel. I give them a URL, and they can do the rest themselves.

You can use any web server for this purpose. I used the built-in Apache server in Mac OS X Server, but you could just as easily use the Microsoft IIS web server with or without Active Directory, or whichever server/authentication mechanism you prefer.

USING THE **OTA SYSTEM**

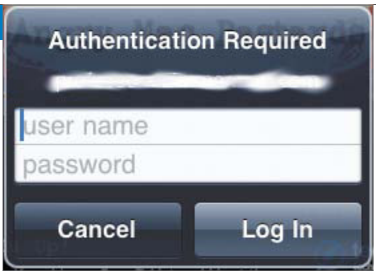


FIGURE 8.2 Authentication dialog for the profile distribution website

So how does this setup work in the real world? Pretty well as it turns out. Since the website is accessible everywhere, you can install the profile using Wi-Fi, 3G, or whatever. The tech doesn't matter as long as the device has Internet access. When my own phone was stolen, I installed the profile I needed while walking from the Apple Store to my hotel.

The user enters the URL in Safari on the iOS device and waits a few seconds for the site to connect. Once the connection is established, he fills in a login dialog with a user name and password, the same login he uses with his computer or into email (**Figure 8.2**).

A few seconds after tapping Log In, he's viewing an Install Profile screen on the iOS device. He taps Install, enters some basic information such as an email address and password (again), and about a minute later, his iOS device has the Wi-Fi network password, the Exchange Activesync account, and our internal certificates installed and ready to go.

It really is that simple. Using this system has also given me a nice bit of flexibility. For example, we have one group that travels a lot, but not all members of that group have their own iPhone, either personal or company-issued. However, we have a "pool" of two or three iPhones that they can sign out. Because we don't want to completely tear down the profiles and reinstall them every time a device is checked out, I created two profiles for these phones.

The first profile, which I installed with iPCU via USB tethering, contains basics such as the Wi-Fi network info, internal certificates, and so on. This profile requires a password to uninstall.

I then uploaded a separate config profile that sets up an Exchange Activesync account on the device. When someone checks out the phone, she installs that profile and has access to all her email and calendaring on that device while she travels. Before the user turns in the device, she deletes that profile and all the personal info is gone.

This method simplifies the configuration process immensely. Most importantly, as long as the server is up and contains the file, the process requires no IT support. That's better for the people using the devices, and it's better for me. Win-win, indeed.

If you need to update the profile, that's no great pain either. Edit the same profile using the iPCU. If you're using Profile Manager in Mac OS X Server 10.7, make the changes to the profile you want and then re-download it. (Although, if you've already installed the profile on the device, you'd be better off just letting Profile Manager push the changes out to the device. But, you can update profiles this way with Profile Manager.) Don't change the profile identifier, and if you sign the profile, make sure to modify it running the same copy of the iPCU on the same computer on which you created the profile. Export the profile to a .mobileconfig file and upload it to the web server, replacing the old version. Then have your users install the modified profile. The new profile will overwrite the old profile's settings with no need to uninstall and reinstall the profile.

DISTRIBUTING **APPLICATIONS OTA**

So you can configure devices wirelessly, but what about distributing apps? Surely for that, you'd need a tethered connection, right? Nope, not at all. In fact, Apple tells you exactly how to do this in the document http://developer.apple.com/library/ios/#featuredarticles/FA_Wireless_Enterprise_App_Distribution/Introduction/Introduction.html, titled "Distributing Enterprise Apps for iOS 4 Devices."

However, the initial setup for the app and the provisioning profile is a little more complicated than for a configuration profile.

First, you have to share the app and distribute for enterprise in Xcode to create an .ipa archive of the app, and a .plist file that is the Wireless Manifest File. You then create a URL that links to the .plist file, similar to this:

```
<a href="itms-services://?action=download-manifest&url=
→http://example.com/manifest.plist">Install App</ a>
```

Note that even though the URL references "itms-services," the iTunes Music Store is not actually involved. Because .ipa and .plist files are not typical web server files, you may have to add the correct mime types to your server. For the .ipa file, the MIME type is application/octet-stream; for the .plist file, the MIME type is text/xml.

You'll also need to make sure that the devices you want to receive applications can contact two specific Apple websites. The first site is ax.init.itunes.apple.com, which is used to handle file-size limitations when downloading files over cellular networks. If you've ever tried to install really big apps from the App Store over 3G and were told the app was too big to install via 3G and requires Wi-Fi, that's what this website is for.

The second site is ocsp.apple.com, which validates the distribution certificate for the app.

If the iOS device cannot contact both of these sites, the user may not be able to install or run the app. (Many thanks to @mdhughes for his help with this section. I really do have to make a big shoutout to my Twitter compatriots in general. This book would have been much harder to write without them.)

WRAPPING **UP**

So not only can you set up a really simple OTA system to configure phones, you can use that same system to distribute apps to your iOS devices. In the next few chapters, we'll enhance OTA services in ways that are more complicated than our "just use a webserver" model, which will provide far more capability and flexibility across the entire spectrum of iOS device management.

NOTE: You'll find a LOT more information on wireless distribution in Chapter 17.

9

SCEP: A BACKGROUND

So far, we haven't really talked much about certificates and profiles beyond installing them on a device, and how you need them for enterprise application distribution. We're going to change that, because although certs and profiles can be some of the biggest pain points an IT administrator faces, they're also *really* useful, and Apple has done quite a bit of work to make adding them into your iOS device much easier.

ENTER **SCEP**

All the installation methods we've looked at thus far have limitations that we've not been able to reconcile. When you want to encrypt your profiles to include passwords or internal certificates that you don't want installed on just any device, you've had to physically connect your iOS device to a computer running the iPhone Configuration Utility (iPCU), or create a separate configuration profile for every device after physically connecting that device to a computer running iPCU.

That's okay for small deployments, or when you have devices trickling into your company in small numbers. But what about large deployments, or when you must ready a huge numbers of devices in a hurry? We currently have solutions that scale, but don't allow us to safely include passwords or proprietary information. Or, we have solutions that are encrypted to include passwords and proprietary information, but they don't scale for beans.

There's got to be a better way, and there is: SCEP, or the Simple Certificate Enrollment Protocol.

SCEP predates the iOS by quite a few years, and was designed to help sysadmins handle SSL Certificate Management in a way that was scalable, reliable, secure, and usable by any valid network user without the need for IT support. It was created by Cisco as, among other things, a way to make it much easier to enroll in certificate-based VPNs (something Cisco has a bit of an interest in).

The (very!) basic idea with SCEP is that once you have some valid form of network credentials, usually LDAP-based in a modern network, you can log onto a web site with your device, and sign in using those credentials. From there, your device and the web site identify themselves to each other, and you can get a certificate to access VPNs or anything else that requires a certificate.

Note that the device identifies itself to the web site. As part of encrypting a profile, the device must be identified to the iPCU so that the device's key can be used to encrypt the profile. This ensures that only that specific device can unencrypt and use the profile. That's secure; but when you're trying to hand out these profiles on a large scale, that's painful. SCEP is a way to deal with that pain. (Although as we'll see in later chapters, SCEP itself can be quite painful.)

For iOS, Apple figured out a way to integrate SCEP with configuration profile distribution so that you could create and install profiles for devices that were secure, encrypted, distributed entirely OTA, and didn't require any IT intervention. As long as you have valid network credentials, you can get the right profile for your iOS device, and it can even have your username, password, email address, and all the rest. Even better, you don't have to go to a URL that references a specific .mobileconfig file. You can just go to http://scepserver.mycompany.com/ and SCEP, plus a few other tricks, takes care of the rest.

"A **FEW** OTHER **TRICKS**"

So about that "SCEP, plus a few other tricks..." thing. I think the biggest problem with SCEP (one that Apple has done a really terrible job explaining) is that SCEP is *only* there to deal with secure certificate distribution, a.k.a. enrollment. As we'll see, all the rest of the process happens completely outside of SCEP. But, because of the (minimal, but slowly getting better) documentation on OTA iOS enrollment , it is *very* easy to think that SCEP actually handles the entire process. It doesn't, but thinking it does will lead you deep down the rabbit hole. I know that it caused me some major headaches, as I was conceptually way out in left field about what was happening.

CONFIGURING iOS DEVICES VIA SCEP

So let's look at what's really going on at a high level. (This is sourced directly from Apple's documentation at http://developer.apple.com/library/ios/featuredarticles/FA_Wireless_Enterprise_App_Distribution/Introduction/Introduction.html#//apple_ref/doc/uid/TP40009979. Gotta love Apple URLs.) There are three major phases to OTA profile delivery using SCEP:

1. Authentication
2. Certificate enrollment
3. Device configuration and encrypted profiles

AUTHENTICATION

The authentication phase is basically what you set up in Chapter 8. You hand out a URL to the user, via email, SMS, or what have you. She goes to that URL and authenticates to it. (For our purposes, we're going to assume you have an LDAP-based Directory Service handling your user authentication data, but it's not required.) You can even check against a list of approved devices, so if someone manages to suss out the authentication credentials but doesn't have the right device, they're shut out.

Once this is done, the site loads a minimal configuration profile onto the device, which contains a request for certain device-specific information to be used in later steps. This can include the iOS version, the MAC address of the Wi-Fi interface, the product type (such as iPhone 2.1), the IMEI number (if applicable), and/or the SIM card identifier number, or ICCID (if present).

The device returns this information to the server via HTTP POST. (In case it needs mentioning, *clearly* this web site should *only* be accessible as an SSL link.) This response is signed by the device with its built-in key via an internal certificate from Apple. So we now have the information we need to start using an encrypted profile.

So far, all the user has had to do is log in to a web site. You can, if you'd like, create a challenge of some kind that the user has to respond to and use that as an additional layer of security. The challenge can also be specific to the device if you like, such as UUID, IMEI, or MAC address. (If you're going to use a device-specific bit of info, you'll need to have it ahead of time, so that's probably not the best strategy if you're allowing people to enroll their personal devices.)

CERTIFICATE ENROLLMENT

The server then sends back a *second* configuration profile with the information needed to make use of SCEP. This profile installation is done in the background; the user doesn't need to know about it at all. Obviously, this profile should be signed by the server. It contains the infomation needed to create a certificate signing request, or CSR.

A QUICK AND (VERY) DIRTY **CSR EXPLANATION**

When you create an SSL certificate, you first need to create a request. This request, or CSR, contains information such as the DNS name of the server (if you're getting the cert for, say, a web site), the company name, the location of the company, the encryption key size, and so on. The information in the CSR can vary depending on need. Basically, the idea is to use enough unique information so that a proper certificate can be generated that identifies a web site, or a device, or even a person. The CSR is then used by the certificate authority to create the certificate you actually use.

For iOS devices, the profile provides information such as company name, the key type, and the UUID number. An example of such a profile is available from Apple, and it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
→"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>PayloadVersion</key>
        <integer>1</integer>
        <key>PayloadUUID</key>
        <string>Ignored</string>
        <key>PayloadType</key>
        <string>Configuration</string>
        <key>PayloadIdentifier</key>
        <string>Ignored</string>
        <key>PayloadContent</key>
        <array>
            <dict>
                <key>PayloadContent</key>
                <dict>
                    <key>URL</key>
                    <string>https://scep.example.com/scep</string>
                    <key>Name</key>
                    <string>EnrollmentCAInstance</string>
                    <key>Subject</key>
                    <array>
                        <array>
                            <array>
                                <string>O</string>
                                <string>Example, Inc.</string>
                            </array>
                        </array>
                        <array>
                            <array>
                                <string>CN</string>
                                <string>User Device Cert</string>
                            </array>
                        </array>
                    </array>
                    <key>Challenge</key>
                    <string>...</string>
                    <key>Keysize</key>
                    <integer>1024</integer>
                    <key>Key Type</key>
                    <string>RSA</string>
                    <key>Key Usage</key>
                    <integer>5</integer>
                </dict>
                <key>PayloadDescription</key>
                <string>Provides device encryption identity</string>
                <key>PayloadUUID</key>
                <string>fd8a6b9e-0fed-406f-9571-8ec98722b713</string>
                <key>PayloadType</key>
                <string>com.apple.security.scep</string>
```

```
            <key>PayloadDisplayName</key>
            <string>Encryption Identity</string>
            <key>PayloadVersion</key>
            <integer>1</integer>
            <key>PayloadOrganization</key>
            <string>Example, Inc.</string>
            <key>PayloadIdentifier</key>
            <string>com.example.profileservice.scep</string>
        </dict>
    </array>
</dict>
</plist>
```

BIG SCARY CERT **WARNING**

If you are just completely confused by all this talk of certs and CSRs, I highly, *highly* recommend that you stop reading right now. Run, do not walk, to learn about how all this is supposed to work. Certs and SSL and the like are common ways to make things secure, but they are terribly, almost frighteningly easy to get wrong. Even something that seems minor can create a huge gaping hole in what you think is a secure setup. Certificates and the concepts behind them are necessary, but they're not the easiest things to learn. Again, entire books are written about this topic, and if you're dealing with it for the first time, please, please, *please* stop, and either learn what you need to learn, or employ the services of someone who already understands it.

Remember the SCEP server settings in configuration profiles that we talked about in Chapter 5? Here's where they come into play, and if you compare the contents of the plist file above to the settings in the SCEP section of the iPhone Configuration Utility's Configuration Profile setup, you'll see that they match rather nicely. Once this second configuration profile has been installed, the device then contacts the actual SCEP server with the CSR it now has, and receives a device-specific certificate from the SCEP server.

DEVICE CONFIGURATION AND ENCRYPTED PROFILES

Okay, head hurt yet? No? Lucky you, mine does. So, now, we have a cert from the SCEP server, and the profile server that we initially talked to pages ago has the response from our device that was signed with the device's internal Apple-provided cert. This response also provided the configuration server with something rather important to this step: the device's public key. This is the key that will be used to encrypt the *third* profile we're about to install. By using the device's *public* key, we ensure that the profile can be decrypted only with the device's *private* key.

PRIVATE AND **PUBLIC KEYS**

Okay, if you thought the basics of SSL were brain-bending, this entire "keys" thing is just as bad, maybe worse. It's all part of a concept called PKI, or Public Key Infrastructure. Basically, it is this. You have a public key and a private key. You give out the public key willy-nilly—the willy-nilly-er the better. You never, never, *ever* give out or allow access to the private key.

So let's say you have a message that you want to send, and you really want people to *know* you and only you sent it. Once you have composed the message, you sign it with your private key and send it. People can then use your *public* key to verify that you signed that message. (When you hear people talking about digital signatures, this is what they should be talking about, not silliness involving scans of their pen and ink signatures.) Conversely, if someone wants to send you something, and ensure that only you can read it, he can encrypt it with your *public* key, and only you can decrypt it with your *private* key.

Confused yet? It gets better. Someone can also give you his public key, and after signing that message with *your* public key, he can then sign it with his *private* key. So, now you get an encrypted, signed message. You use his public key to verify the sender, and use your private key to decrypt it so you can actually see the message. Not only are you (in theory) the only person in the world who could read this message, but you are sure (in theory) of who sent the message. If you really want to get a better understanding of this, I highly recommend reading the document at www.carillon.ca/library/enrolment_1.1.pdf?page=tutorials. Yes, it is in fact called "PKI Enrollment A Fingerpuppet-Theatre Guide". It is also a well-done, accurate explanation of PKI, and why it's useful. You really need to understand PKI for this chapter to be as useful to you as it can be.

The device sends a signed request for the final profile to the profile server. The request is signed with the certificate the device received from the SCEP server. The profile server creates the final profile, encrypts it with the device's public key, and sends this encrypted profile to the device. The device receives, decrypts, and installs the profile. The device is now configured.

WRAPPING **UP**

So with only two servers, three configuration profiles, certificates, CSRs, public and private keys galore, we have a configured device. You all may be wondering: “Why, oh why, would I *ever* do this? Is it that much to ask people to enter in a user ID and password?” Well, probably not. But let’s take a look at this.

Without SCEP, you can’t easily encrypt the profiles. So you can only automate the configuration process so far. You could automate the profile creation to include passwords; but without encrypting the profile, it’s a really bad idea.

SCEP helps you encrypt your profiles without manually tethering the device to a Mac or Windows box running the iPhone Configuration Utility. With SCEP, you can further automate the processes we’ve talked about in earlier chapters and do it in a more secure manner. So, iOS device setup becomes even more convenient for users, because if you take advantage of profile creation automation and SCEP’s security, you can integrate profile creation into your infrastructure. Your users can get their devices configured as needed without spending a lot of time entering what is essentially duplicate information. And you don’t have to spend a lot of time creating custom profiles, or attaching devices to a computer (or three). Plus, it’s all done fairly securely. What could possibly be the problem?

As we’ll see, the problem is that setting up all this, especially SCEP, is a lot harder than it should be unless you use one of the many commercial implementations available.

This page intentionally left blank

10

IMPLEMENTING **SCEP** ON **MAC OS X SERVER**

In Chapter 9, we talked about the basics of SCEP and the workflow to use with SCEP, as well as the whats and whys of SCEP. In this chapter, we'll look at implementing SCEP and the associated workflow in Mac OS X Server. Like the rest of the book, this isn't going to be a "step one, do this, step two, do that" kind of chapter. Rather, we'll examine some of the issues involved, explore ways to handle the problems that tend to crop up, and check out some of the implementations of SCEP available for Mac OS X Server.

SETTING UP **SCEP** ON **MAC OS X SERVER**

Let's start with the Apple-provided implementation of SCEP. The way you set up SCEP on Mac OS X Server differs wildly depending on which server version you're using. If you're on 10.6 or earlier, the story is a little complicated. If you're on 10.7, well, things get a lot simpler.

IMPLEMENTING SCEP ON MAC OS X 10.6 SERVER

So, what's the best way to set up Apple's SCEP implementation on Mac OS X10.6 or earlier? That's easy. There isn't one. So now, on to…hmm? No, I wasn't kidding. Through Mac OS X 10.6 Server, Apple provided no SCEP implementation that installs and integrates with other services in Mac OS X Server. Go check their downloads, and check out Mac OS X Server. It's not there. Read this from the "Configuring the Infrastructure" section in the "Over-the-Air Profile Delivery and Configuration" at www.apple.com/ipad/business/resources/:

> Certificate Services
>
> The process of enrollment requires deployment of standard x.509 identity certificates to iOS users. To do this, you will need a CA (certificate authority) to issue the device credentials using the Simple Certificate Enrollment Protocol (SCEP).
>
> Cisco IOS and Microsoft Server 2003 (with the add-on for certificate services) both support SCEP. There are also a number of hosted PKI services that support SCEP, such as Verisign, Entrust, and RSA. For links to PKI, SCEP, and related topics read the "See Also" section in "Introduction."

If Apple had a SCEP solution that you just installed, they might, you know, mention it here. Or anywhere else. (I didn't conclude this only from the documentation referenced above. I asked quite a few people at Apple, who shall remain nameless, and the responses were variations on "No, darnit" and, "I wish!"

So you aren't getting this solution from Apple. An open source SCEP server, OpenSCEP, is available from http://openscep.othello.ch/. As near as I can tell, it won't build on Mac OS X Server. To be fair, I stopped trying when it told me I didn't have a compiler such as gcc or cc in my PATH variable, when my path very clearly shows /usr/bin, which contains both gcc and cc. If you just need to make a few adjustments here and there, I'm down with that. But dealing with this level of not acknowledging reality? Not so much. I'm not the only one, the silly thing just doesn't want to build. Then again, it hasn't been updated in a dog's age. As near as I can tell, no one's done anything with it since 2001. $DEITY knows *what* OpenSCEP will build on now.

I asked around quite a bit, and was told that if I wanted to implement a full SCEP-centric solution on Mac OS X, I need to use a commercial product, and I've found no hint that this is incorrect. Fortunately, a lot of SCEP/iOS management packages are out there. For this chapter, we'll look at Casper from JAMF Software, www.jamfsoftware.com. Casper's a solid package, and the folks at JAMF are monstrously cool. They were of no small help when answering my questions about Casper and SCEP/Mobile Device Management, in general, and I'm not even a customer.

I don't want people to think Apple doesn't do anything to help with setting up a SCEP-centric solution. The documents linked from the iPad in Business: Resources page does have some excellent documentation on setting up everything *but* SCEP, including the all-important Open Directory integration with the web server via some clever Ruby sample scripts. However, since we have to talk about something like Casper to talk about SCEP on Mac OS X Server *anyway*, in for a penny, in for a pound I always say. (Well, I would if I were British. And Charles Dickens.)

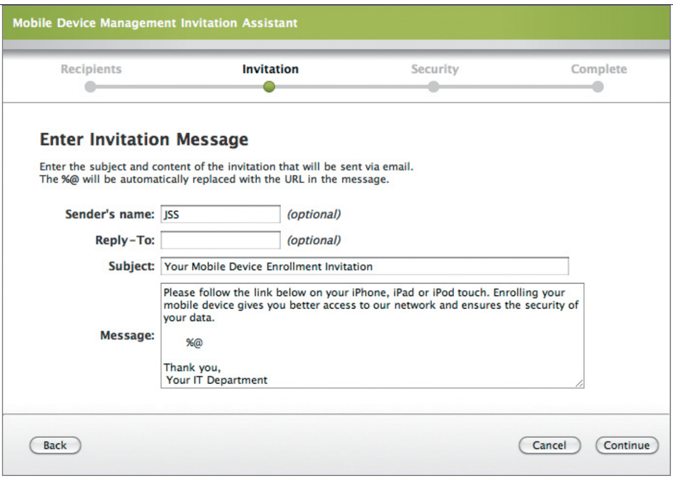
SETTING UP SCEP WITH CASPER

I don't want to give the impression that Casper is the only game in town, or that you're mad if you don't use Casper. A lot of other packages will also manage your iOS devices quite well. Some run on Mac OS, such as Casper and Absolute Manage (Formerly LANRev), while some run on Windows, such as Good and Sybase's Afaria. Others are cloud-based, such as airwatch. But, you can't demo everything (not if you want to finish writing a book), and since I'm getting the gimlet eye from my far-too-patient editors already, we'll use Casper from JAMF Software.

Setting up SCEP in Casper is, as it turns out, pretty easy. You can use your own certificate authority (CA) or the built-in CA included with Casper. Once that's done, you begin creating the multiple profiles you'll need to set up SCEP. Casper requires that you first get an Apple Push Notification Certificate. That process is just a tad... byzantine. For example, it would not occur to me that I'd need to create a new app ID for this because I'm not using it for apps; but evidently, Apple sees it differently. Fortunately, JAMF has a solid tech note on this process at http://jamfsoftware.com/libraries/pdf/white_papers/JAMFSoftware-Creating_an_Apple_Push_Notification_Certificate.pdf. I can say it works because I used it while writing this chapter. (That is, I did exactly what the nice document said and everything worked out okay. That, folks, is good documentation.)

Following this, you'll create the enrollment profile. That's pretty simple as it's a "just enough to care" profile. Then, you have a few options for enrolling users. You can do this completely over the air (OTA), you can send users a URL that they can go to for enrollment, or you can place the enrollment profile into a copy of the iPhone Configuration Utility and install it that way. We, of course, will look at OTA because the idea is to make this as simple as possible for everyone, and OTA does that.

Casper has a rather nice wizard to set this up (**Figure 10.1**), and allows you to send the invitations via SMS message or email. When those are sent, a few minutes or seconds later, users get a text message or email with a URL. Then they tap the URL to start the enrollment (**Figures 10.2** and **10.3**).

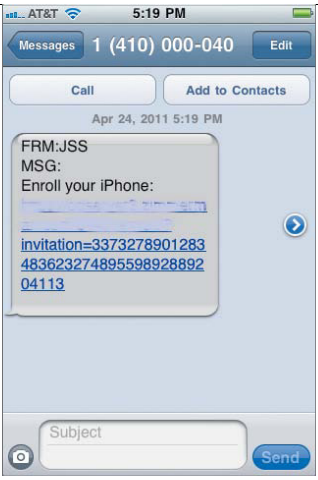


Enter Invitation Message

Enter the subject and content of the invitation that will be sent via email.
The %@ will be automatically replaced with the URL in the message.

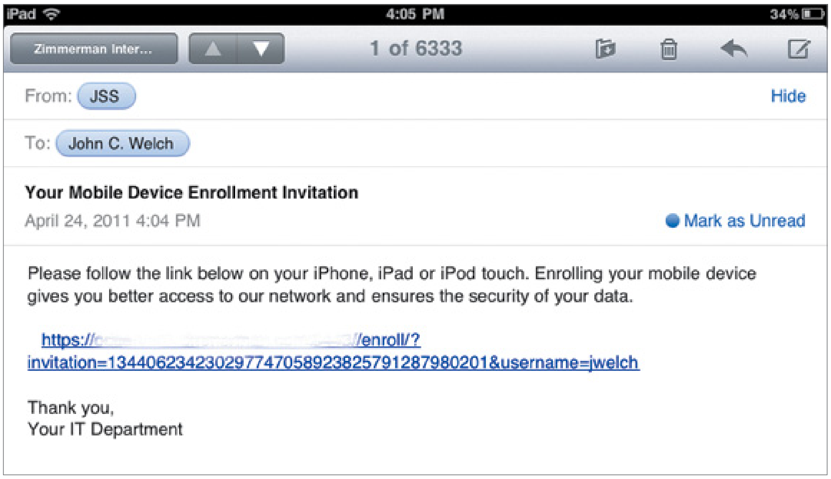
Sender's name:	JSS	(optional)
Reply-To:		(optional)
Subject:	Your Mobile Device Enrollment Invitation	
Message:	Please follow the link below on your iPhone, iPad or iPod touch. Enrolling your mobile device gives you better access to our network and ensures the security of your data. %@ Thank you, Your IT Department	

FIGURE 10.1 OTA Invitation wizard



FRM:JSS
MSG:
Enroll your iPhone:
[illegible]
invitation=3373278901283
483623274895598928892
04113

FIGURE 10.2 SMS invitation



From: JSS

To: John C. Welch

Your Mobile Device Enrollment Invitation

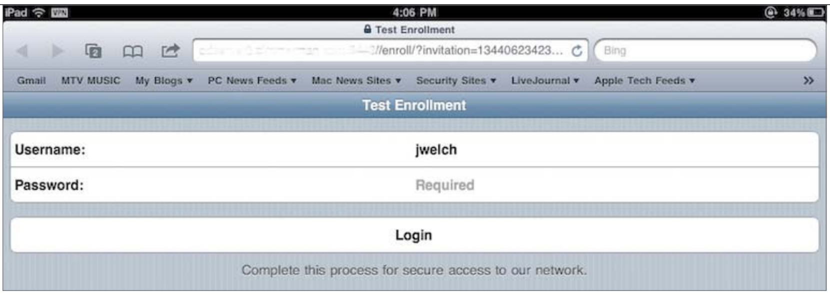
April 24, 2011 4:04 PM

Please follow the link below on your iPhone, iPad or iPod touch. Enrolling your mobile device gives you better access to our network and ensures the security of your data.

https://[illegible]/enroll/?
invitation=13440623423029774705892382579128798020l&username=jwelch

Thank you,
Your IT Department

FIGURE 10.3 Email invitation



Username: jwelch

Password: Required

Complete this process for secure access to our network.

FIGURE 10.4 LDAP-based authentication to the enrollment website

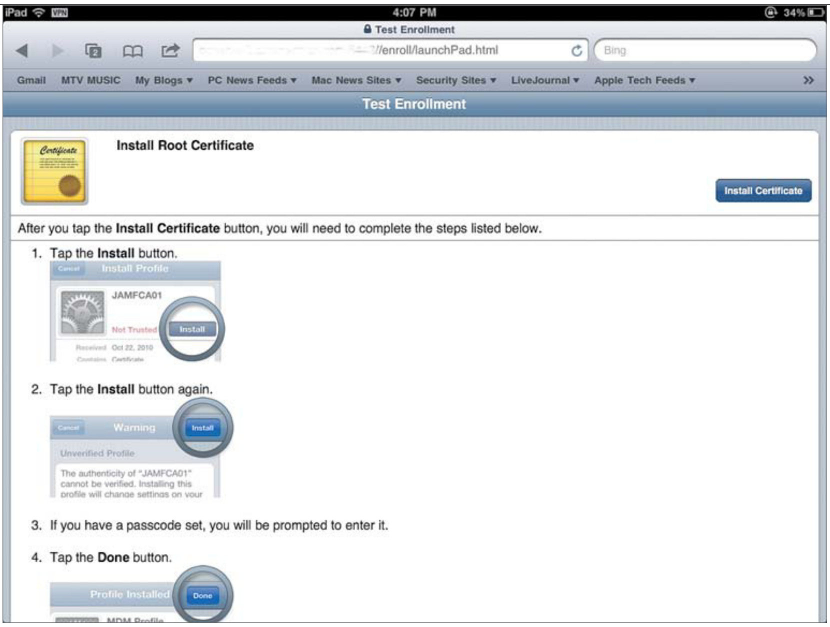
One handy thing about Casper (and other management packages) is that you can integrate your enrollment process with your LDAP directory. So, just as we did manually in Chapter 8, you can ensure that someone who gets the SMS or email by mistake can't enroll a random device (**Figure 10.4**).

Once you log in, Casper does something I find quite nice: It displays a page that tells the user what's about to happen. Casper does this at every stage of the enrollment process in a way that can be understood by non-IT people. If you are going to roll your own enrollment web site, you might want to think about doing the same thing. It's a little more effort when setting up your infrastructure, but your users will really appreciate it.

First, you're prompted to install the root certificate authority that will act as the basis for the other certs you'll use in this process (**Figure 10.5**). (Notice that it's the same steps you would take to install any other kind of cert or profile.)

You're then asked to install the Mobile Device Management profile and certificate because Casper makes use of the fine-grained control you get with Mobile Device Management over generic configuration profiles (**Figure 10.6**). (We cover Mobile Device Management in depth starting with Chapter 13.)

Once that's installed, the user is done and you, as the IT person, can start pushing out configurations to the devices. One thing to keep in mind is that even though our example suite, Casper, uses Mobile Device Management more than traditional configuration profiles, if you do distribute traditional configuration profiles, even encrypted ones, you're still bound by their limitations. That is, when you want to change anything, you have to push out an entirely new configuration profile with your changes. This works, and face it, configuration profiles aren't exactly huge; but, as we'll see in the Mobile Device Management chapters, you have a much better way to deal with configuration management than the "all or nothing" approach of configuration profiles.



Install Root Certificate

Install Certificate

After you tap the **Install Certificate** button, you will need to complete the steps listed below.

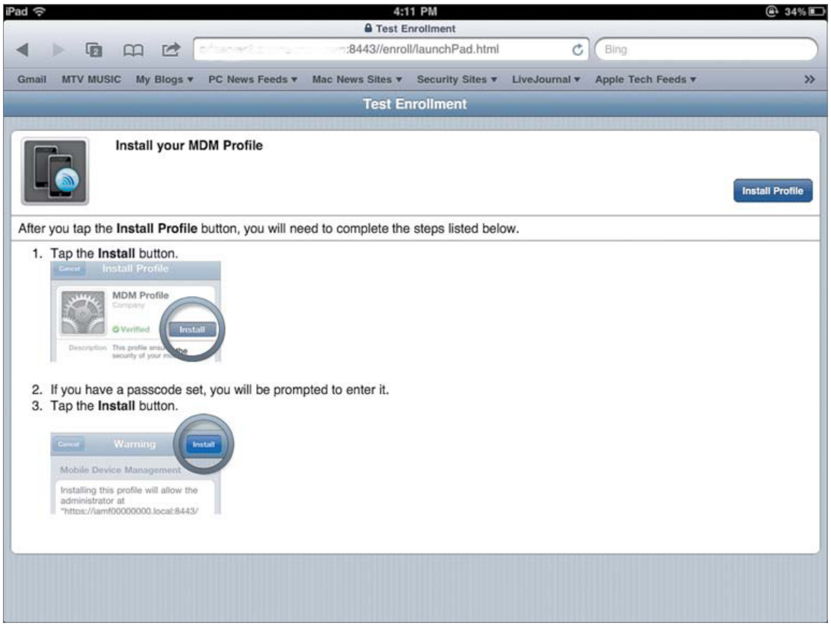
1. Tap the **Install** button.

2. Tap the **Install** button again.

The authenticity of "JAMFCA01" cannot be verified. Installing this profile will change settings on your

3. If you have a passcode set, you will be prompted to enter it.
4. Tap the **Done** button.

FIGURE 10.5 Root Enrollment prompts



Install your MDM Profile

Install Profile

After you tap the **Install Profile** button, you will need to complete the steps listed below.

1. Tap the **Install** button.

2. If you have a passcode set, you will be prompted to enter it.
3. Tap the **Install** button.

Installing this profile will allow the administrator at "https://jamf00000000.local:8443/

FIGURE 10.6 MDM Enrollment prompts

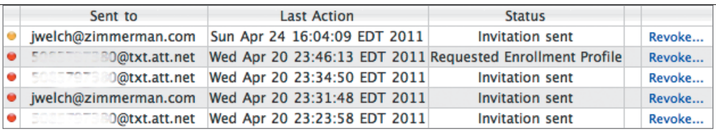


FIGURE 10.7 Invitation sent status

	Sent to	Last Action	Status		
🟠	jwelch@zimmerman.com	Sun Apr 24 16:04:09 EDT 2011	Invitation sent		Revoke...
🔴	[redacted]0@txt.att.net	Wed Apr 20 23:46:13 EDT 2011	Requested Enrollment Profile		Revoke...
🔴	[redacted]0@txt.att.net	Wed Apr 20 23:34:50 EDT 2011	Invitation sent		Revoke...
🔴	jwelch@zimmerman.com	Wed Apr 20 23:31:48 EDT 2011	Invitation sent		Revoke...
🔴	[redacted]0@txt.att.net	Wed Apr 20 23:23:58 EDT 2011	Invitation sent		Revoke...

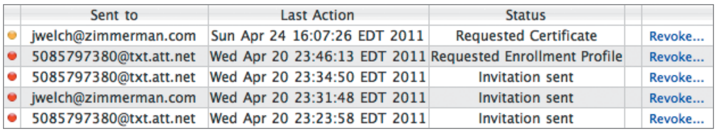


FIGURE 10.8 Enrollment in progress status

	Sent to	Last Action	Status		
🟠	jwelch@zimmerman.com	Sun Apr 24 16:07:26 EDT 2011	Requested Certificate		Revoke...
🔴	5085797380@txt.att.net	Wed Apr 20 23:46:13 EDT 2011	Requested Enrollment Profile		Revoke...
🔴	5085797380@txt.att.net	Wed Apr 20 23:34:50 EDT 2011	Invitation sent		Revoke...
🔴	jwelch@zimmerman.com	Wed Apr 20 23:31:48 EDT 2011	Invitation sent		Revoke...
🔴	5085797380@txt.att.net	Wed Apr 20 23:23:58 EDT 2011	Invitation sent		Revoke...

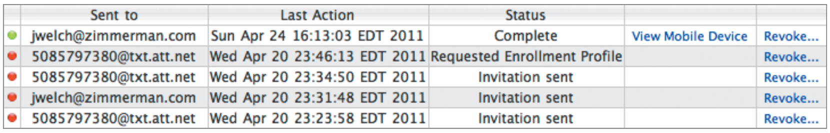


FIGURE 10.9 Enrollment complete status

	Sent to	Last Action	Status		
🟢	jwelch@zimmerman.com	Sun Apr 24 16:13:03 EDT 2011	Complete	View Mobile Device	Revoke...
🔴	5085797380@txt.att.net	Wed Apr 20 23:46:13 EDT 2011	Requested Enrollment Profile		Revoke...
🔴	5085797380@txt.att.net	Wed Apr 20 23:34:50 EDT 2011	Invitation sent		Revoke...
🔴	jwelch@zimmerman.com	Wed Apr 20 23:31:48 EDT 2011	Invitation sent		Revoke...
🔴	5085797380@txt.att.net	Wed Apr 20 23:23:58 EDT 2011	Invitation sent		Revoke...

I want to touch on one other service that Casper provides: progress indicators. One of the problems you're going to have when enrolling and configuring devices is determining if the people you sent the invitations to ever accepted them. There's all kinds of reasons why they might not have gone through the process, but if you can't tell that from your end, you're reduced to visually inspect the device to see if it happened (and if you do that, why bother with OTA at all), or sending out repeated emails asking "Have you enrolled yet?" Neither of these is a great option. Casper helps you with a status display that shows the stages of enrollment for each device. So you can tell when an invitation has been sent (**Figure 10.7**), when the device has requested the root certificate or the enrollment profile (**Figure 10.8**), and when a device has completed enrollment, (**Figure 10.9**).

If you have a large number of devices in geographically disparate locations, this kind of easy to read and monitor status display is invaluable. (As are the handy "revoke" options for each invitation, in case someone's device is lost or stolen before it can be enrolled.)

IMPLEMENTING **SCEP** ON **MAC OS X SERVER 10.7**

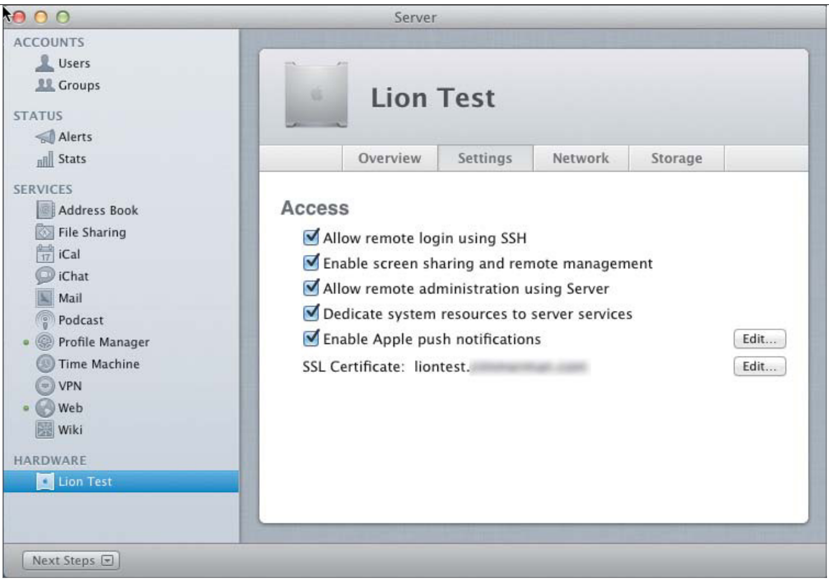
Aside from the low cost premium ($49.00 US on top of the cost of Mac OS X 10.7), there's not much to implementing SCEP on Mac OS X Server 10.7. You will need to set up a few things ahead of time. First, the server you're going to use for this really, really needs a static IP address. Servers in general don't like IP addresses that change, and Mac OS X Server gets particularly flustered and stern about this.

Secondly, you have to have a good DNS for this server. That means a proper server name, such as "iosserver.bynkii.com," tied to a static IP address. You have to have functioning A records (that translate the DNS name to an IP address) *and* PTR records (that translate the IP address to a DNS name). If those are not set up correctly, you will have a heck of a time getting Mac OS X Server 10.7 to work correctly at all, much less managing iOS devices.

Having proper DNS and a static IP address for servers is a good idea in any case, but if you don't set that up correctly for Mac OS X Server, you will be in a world of pain.

Third, you want to have a "real" SSL cert from a "proper" issuing authority—Verisign, GoDaddy, Comodo, whatever. You can, in theory, use self-signed certificates for SCEP, and Casper does it rather nicely. However, if you are not extremely careful with your SSL setup, self-signed certs are going to cause you a plenty of hurt. Given that you can get a cert from a "proper" authority for as little as $50 a year, it's a pretty easy decision to make. In my case, I went with GoDaddy, and that's the cert I used for all my testing and the writing of this book.

Before you start setting up the Mac OS X Server 10.7 Profile Manager service (which handles, among other things, SCEP and iOS device management), you'll need to install and set up a certificate for the server. Doing this in 10.7 is fairly simple. First, you generate a CSR, or certificate signing request. You do this by going into the server application and logging into the Mac OS X Server 10.7 server you're using for iOS device management.



Access

- ✓ Allow remote login using SSH
- ✓ Enable screen sharing and remote management
- ✓ Allow remote administration using Server
- ✓ Dedicate system resources to server services
- ✓ Enable Apple push notifications

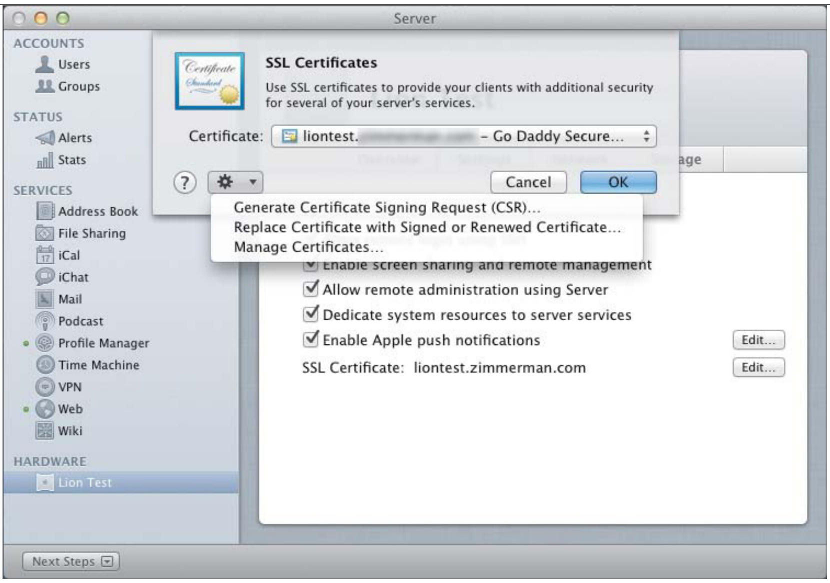
SSL Certificate: liontest.[redacted]

FIGURE 10.10 Hardware settings

In the hardware section, select that server, and click the Settings tab (**Figure 10.10**) Click the Edit button on the SSL Certificate line, and then click the Settings (gear) button and choose Generate Certificate Signing Request (CSR) (**Figure 10.11**). This will generate the CSR you'll need to submit to your cert provider.

Once you get the certificate (most likely as a .crt file), you'll need to install it, and any optional intermediate certs, into the system. Double-click the files to open the Keychain Access application which manages certificates for OS X. (If you get an intermediate cert, install that first.) Make sure you install the certs into the System keychain.

Then, go back into the Server application to the same place you went to get the CSR. When you click the Certificate pop-up menu, the certificate you just installed should be available. If not, restart the server application (not the entire server mind you, just the application), and return to the SSL management section. It should be available in the pop-up menu (**Figure 10.12**).



SSL Certificates

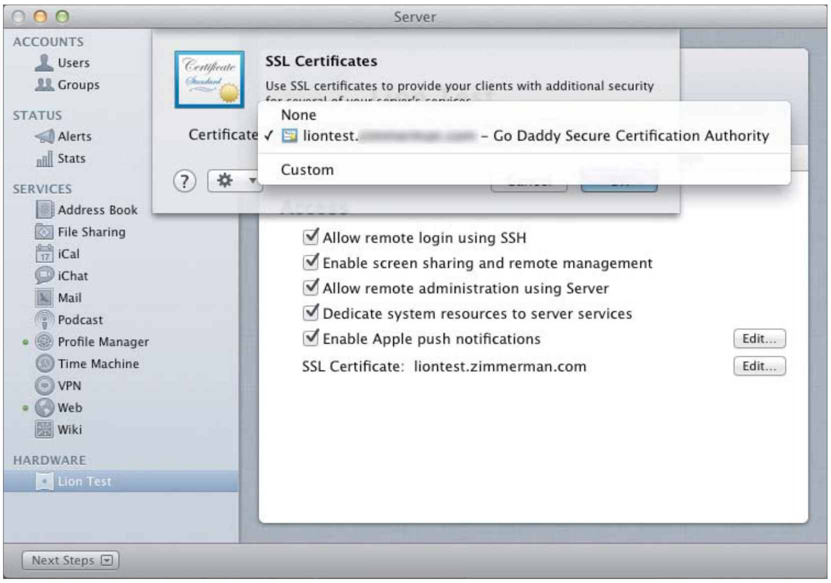
Use SSL certificates to provide your clients with additional security for several of your server's services.

Certificate: liontest.[redacted] – Go Daddy Secure…

✓ Enable screen sharing and remote management
✓ Allow remote administration using Server
✓ Dedicate system resources to server services
✓ Enable Apple push notifications

SSL Certificate: liontest.zimmerman.com

FIGURE 10.11 Server application certificate options



SSL Certificates

Use SSL certificates to provide your clients with additional security

Certificate: ✓ liontest.[redacted] – Go Daddy Secure Certification Authority

✓ Allow remote login using SSH
✓ Enable screen sharing and remote management
✓ Allow remote administration using Server
✓ Dedicate system resources to server services
✓ Enable Apple push notifications

SSL Certificate: liontest.zimmerman.com

FIGURE 10.12 Server certificate selection

If you just select that certificate, you'll be selecting it for use by all services that can use SSL. If you don't want to do that (you may be using other certificates for other services on this server), then choose Custom and make sure that this cert is selected for use by the web server. (If you're not sure whether you need to set up separate certs for each service, you probably don't.)

We're almost done with the preliminaries. The last thing to do is get some push notification certificates from Apple. iOS device management makes heavy use of Apple's Push Notification Service, or APNS, as do the CalDAV, CardDAV, and email services. In Mac OS X Server 10.7, Apple is using the Profile Manger not only for iOS management, but also for user, group, and Mac management. So, you're going to need push certs no matter what. Fortunately, Apple makes it easy to get these. All you need is an Apple ID with an associated email address. (The ID, itself, doesn't have to be an email address, but it needs to have an email address tied to the ID.)

Click the Edit button next to "Enable Apple push notifications," and do what the nice dialogs tell you to do. (It's ridiculously simple.) Once the push certs have been acquired, select the checkbox to enable push notifications, and you're ready to set up Profile Manager.

SETTING UP PROFILE MANAGER

If you have not already set up a Mac OS X Server 10.7-based Open Directory implementation, when you go to enable SCEP on Mac OS X Server 10.7 via Profile Manager, it's going to walk you through setting up that server as an Open Directory Master. This is fairly painless (assuming you have DNS set up correctly), and you don't have to do much more than enter a password for the Directory Administrator, diradmin, login.

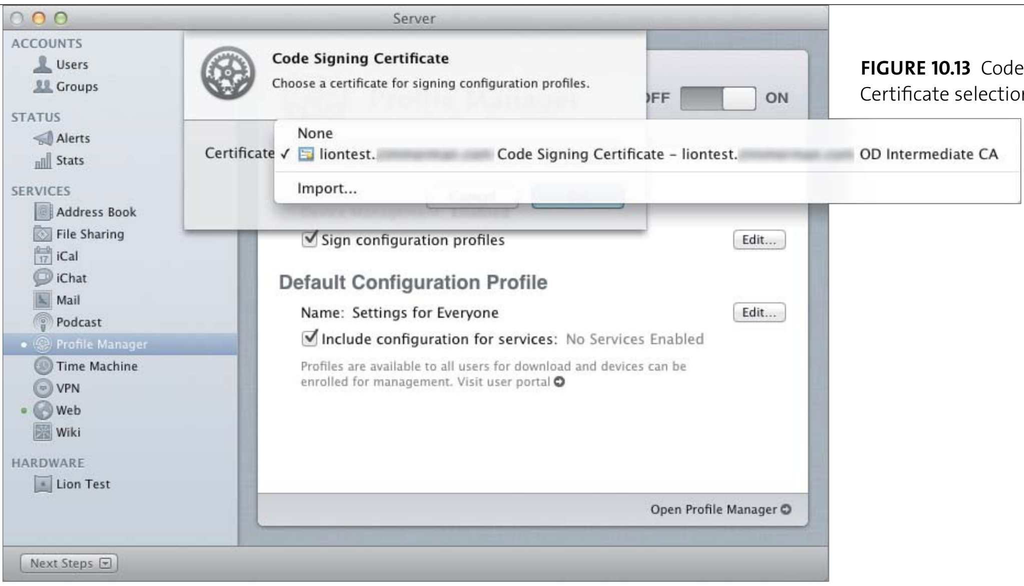

Server
ACCOUNTS
Users
Groups
STATUS
Alerts
Stats
SERVICES
Address Book
File Sharing
iCal
iChat
Mail
Podcast
Profile Manager
Time Machine
VPN
Web
Wiki
HARDWARE
Lion Test
Next Steps
Code Signing Certificate
Choose a certificate for signing configuration profiles.
OFF
ON
Certificate
None
✓ liontest. Code Signing Certificate – liontest. OD Intermediate CA
Import…
Sign configuration profiles
Edit…
Default Configuration Profile
Name: Settings for Everyone
Edit…
Include configuration for services: No Services Enabled
Profiles are available to all users for download and devices can be enrolled for management. Visit user portal
Open Profile Manager

FIGURE 10.13 Code Signing Certificate selection

Once that is done, you'll want to enable signing configuration profiles. Go to the Profile Manager tab in Server, and click the Edit button next to "Sign Configuration Profiles." You'll see another Certificates pop-up menu there. Click it to select the signing cert (**Figure 10.13**). By default, one cert will be generated by Open Directory. Unless you have another signing cert you're using, pick that one, click OK, and then select the checkbox to enable configuration profile signing.

Once that's done, assuming Profile Manger is reading "Device Management: Enabled," all you have to do is turn on Profile Manager, and SCEP will be enabled, along with all the rest of Mac OS X Server 10.7's iOS device management features.

WRAPPING **UP**

And that's the SCEP on Mac story. Obviously, the process becomes easier when using the latest versions of Mac OS X Server, although third-party packages like Casper do bring some benefits to the party. The next chapter talks about setting up SCEP on Windows. As it turns out, the Windows solution is somewhat nicer than the Mac, although it is still not what it should be.

11

IMPLEMENTING **SCEP** ON **WINDOWS SERVER 2008**

In the previous chapter, we implemented SCEP and related services on Mac OS X using Casper from JAMF software. In this chapter, we'll try to do things a little differently by focusing on SCEP without using third-party software.

We are also going to talk about SCEP only in terms of Windows Server 2008. I know a lot of folks are still using Server 2003, but Server 2008 is almost four years old at this point (2011), so I think it's reasonable to focus on the current version.

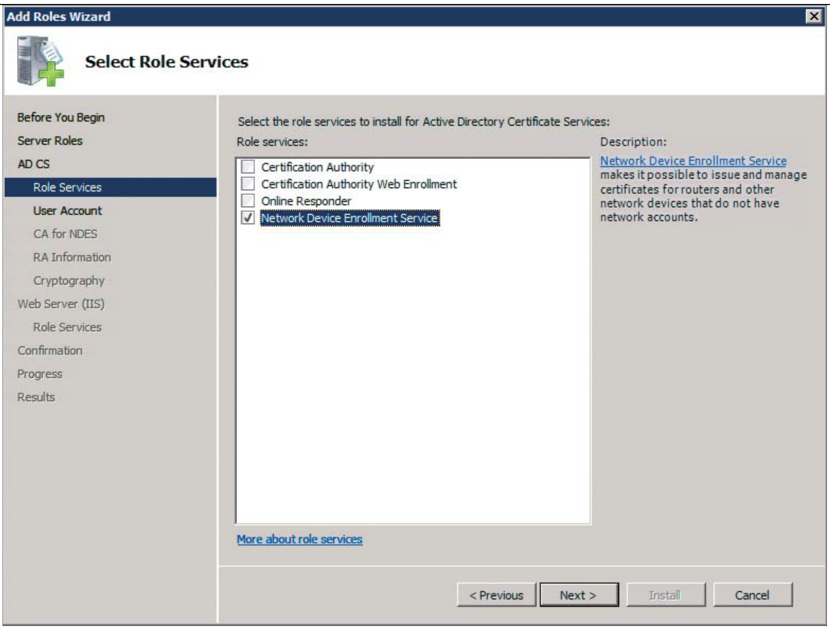
CONFIGURING THE **SERVER**

Using Microsoft servers for all phases of managing iOS devices may not be the easiest thing right now, but using Server 2008 for SCEP is fairly straightforward. Thanks to years of Microsoft support for certificates in the enterprise, the process is fairly well documented (www.microsoft.com/downloads/en/details.aspx?familyid=44315BFF-B744-4637-A66B-E69B4955EE45&displaylang=en and at www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=e11780de-819f-40d7-8b8e-10845bc8d446).

Other individuals have also written awesome articles that you can find at sites like: http://mobilitydojo.net/2010/01/20/sinking-our-teeth-into-scep. This chapter would not exist but for those sites and Microsoft's documentation.

SETTING UP THE ROLES

First, you'll need to make sure your server is set up as both a web and certificate server. This is done using the Server Manager application in Windows Server 2008. You should be running Server 2008 Enterprise or Datacenter for this app to work correctly.



Select Role Services

Select the role services to install for Active Directory Certificate Services:

Role services:

- [ ] Certification Authority
- [ ] Certification Authority Web Enrollment
- [ ] Online Responder
- [✓] Network Device Enrollment Service

Description:

Network Device Enrollment Service makes it possible to issue and manage certificates for routers and other network devices that do not have network accounts.

FIGURE 11.1 Server Role Services

On that server you'll need to be running the Active Directory Certificate Services (ADCS) role installed with the Network Device Enrollment Service (NDES) role service (**Figure 11.1**).

Installing NDES also installs IIS (Internet Information Services) and related support services. If ADCS is not already installed, NDES installation is a little weird because you can't install the Certification Authority and the NDES role services at the same time. First install the Certification Authority and configure the CA, and then install NDES.

NOTE: For this chapter, I'm working with Server 2008 Enterprise in stand-alone mode. If your server is part of an existing Active Directory domain, please, please work with your Active Directory administrators on implementing SCEP. Adding random certificate and device enrollment servers in an existing domain without informing your Active Directory administrators is *bad*.

(If none of this makes any sense to you, you should stop right now, and get in touch with whoever is responsible for Active Directory in your company. This process is not something for the uninitiated to try on a company network, and if you do try this without your Active Directory administrator's cooperation, you'll become his *favorite person in the whole world*. Or not.)

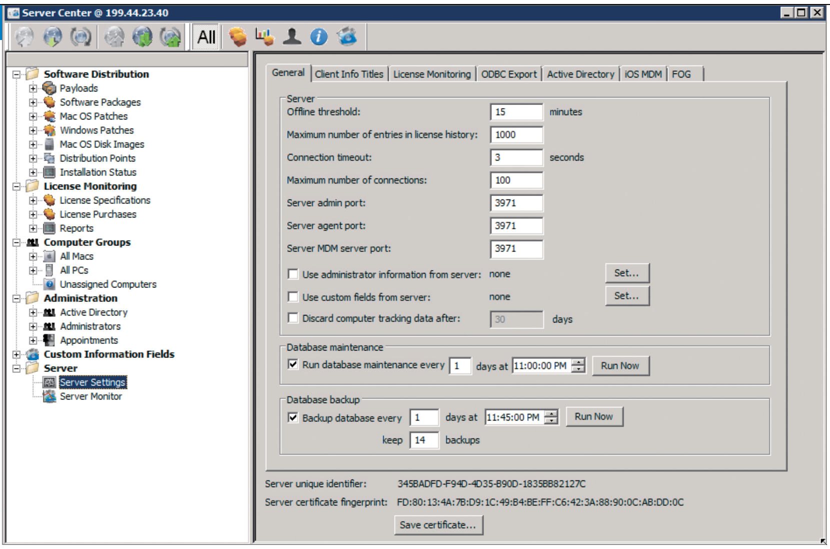
When you get NDES set up, you're basically done. There's only one problem: Getting the server to talk to an iPhone. As it turns out, that's a bit of a mess caused partially by iOS 4.x issuing calls (such as GetCACaps) that aren't well supported by Windows native SCEP.

There are ways around this, but they're all rather painful. So we end up with a well-documented system that just isn't designed to work with iPhones, and it requires a lot of customization to do so. We also end up with a situation where the customizations aren't that well documented.

I don't actually blame Microsoft here. iPhones are not part of their main target market. But, it would be nice to see better documentation since I hear that iOS devices are pretty popular in the Windows world. Since we actually want to get this working, we'll abandon our original intentions and go third-party again, this time with Absolute Manage for Windows.

NOTE: If you're starting to think "Gee, SCEP sounds like a great idea, but there seems to be a lot of pain involved with it, especially if you aren't already an expert," then you would be in complete agreement with me. SCEP is, I think, a great idea. But even the best documentation I've seen for it is incomplete, and the folks doing the documentation for Microsoft/Cisco/Apple and independent sites seem to be writing for themselves, not for people trying to get a handle on using SCEP. One thing writing this book taught me (the hard way) was that there's not yet a good way to just set up SCEP. Going with a product that already has it implemented really is the best option for now.

INSTALLING **ABSOLUTE MANAGE**



Server
Offline threshold: 15 minutes
Maximum number of entries in license history: 1000
Connection timeout: 3 seconds
Maximum number of connections: 100
Server admin port: 3971
Server agent port: 3971
Server MDM server port: 3971
☐ Use administrator information from server: none Set...
☐ Use custom fields from server: none Set...
☐ Discard computer tracking data after: 30 days

Database maintenance
☑ Run database maintenance every 1 days at 11:00:00 PM Run Now

Database backup
☑ Backup database every 1 days at 11:45:00 PM Run Now
keep 14 backups

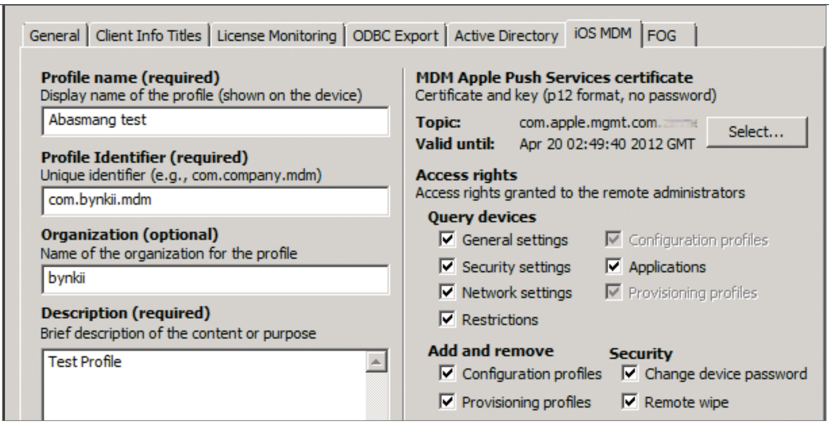
Server unique identifier: 345BADFD-F94D-4D35-B90D-1835BB82127C
Server certificate fingerprint: FD:80:13:4A:7B:D9:1C:49:B4:BE:FF:C6:42:3A:88:90:0C:AB:DD:0C
Save certificate...

FIGURE 11.2 Absolute Manage Server Settings page with certificate export button

The initial install of Absolute Manage is straightforward. You install the server, restart, and then install the management console. The next step is to install the Mobile Device Management server.

As with the Server and the Management console, follow Absolute Manage's instructions because they work well, with only a few places that might bite you. For example, when you get a website certificate, you *have* to include the keys with the cert, and export them as a .pfx bundle. No options. That applies to the first certificate you're asked for. The second one is exported by the Absolute Manage Admin console by clicking the "Save certificate" button in Server Settings in the Server Center (**Figure 11.2**).

NOTE: If you have IIS installed on the same Windows Server 2008 system on which you're installing Absolute Manage, make sure that you install the complete IIS 6 compatibility service roles sets or the Mobile Device Management server won't install.



Profile name (required)
Display name of the profile (shown on the device)
Abasmang test

Profile Identifier (required)
Unique identifier (e.g., com.company.mdm)
com.bynkii.mdm

Organization (optional)
Name of the organization for the profile
bynkii

Description (required)
Brief description of the content or purpose
Test Profile

MDM Apple Push Services certificate
Certificate and key (p12 format, no password)
Topic: com.apple.mgmt.com.
Valid until: Apr 20 02:49:40 2012 GMT

Access rights
Access rights granted to the remote administrators

Query devices
[✓] General settings
[✓] Configuration profiles
[✓] Security settings
[✓] Applications
[✓] Network settings
[✓] Provisioning profiles
[✓] Restrictions

Add and remove
[✓] Configuration profiles
[✓] Provisioning profiles

Security
[✓] Change device password
[✓] Remote wipe

FIGURE 11.3 Absolute Manage iOS Mobile Device Management settings

As with Casper, you'll need to install a push certificate from Apple. You can refer to the documentation referenced in Chapter 10 for doing so. However, when you export the .p12 bundle to import it into Absolute Manage, do *not* add a password. If you do, Absolute Manage won't be able to import it (**Figure 11.3**).

At that point, you should be good to go. Point your device to https://<*server*>/Profile/enrollment.mdm, and you'll be taken to the enrollment page. By default, Absolute Manage ties itself into Active Directory, so anyone going to this site will need a valid login (**Figure 11.4**). After login, the system behaves just like every other enrollment. The user sees the Install Profile screen on her device (**Figure 11.5**), taps the screen a few times, and she's enrolled!

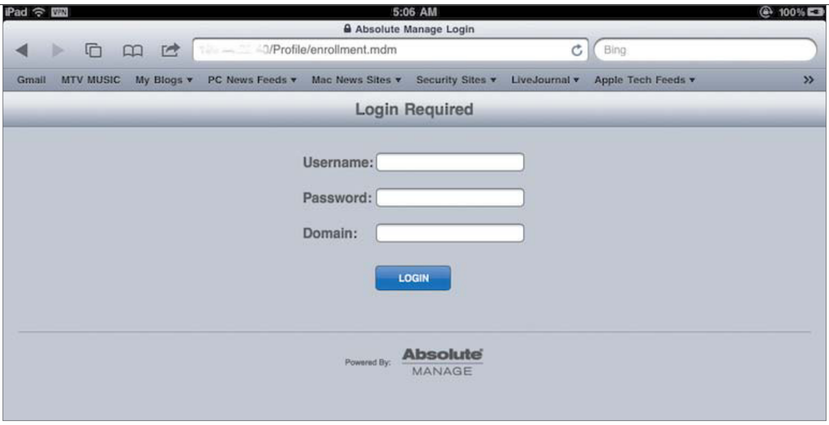
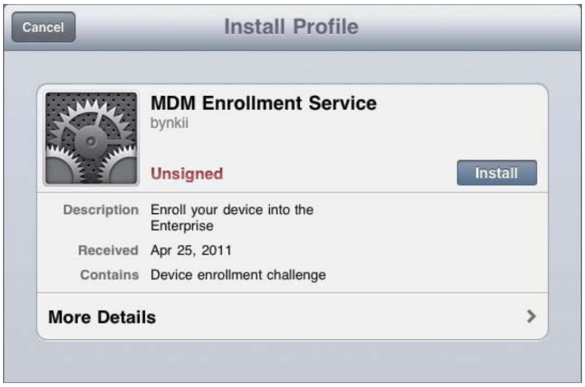


FIGURE 11.4 Absolute Manage enrollment login page



Description	Enroll your device into the Enterprise
Received	Apr 25, 2011
Contains	Device enrollment challenge

FIGURE 11.5 Absolute Manage enrollment profile installation screen

WRAPPING UP

Yes, I know, we still had to use a third party. I'm not really that surprised. If you look at everything that's going on, particularly with SCEP, it's not a simple process. You could roll your own, but in the end, why do what you don't have to? If you have enough iOS devices to justify using SCEP and OTA enrollment, you should recognize that some upfront costs for third-party servers are worth having an easy install and a working system.

There's a reason why so many companies are practically beating down your door to provide nice, packaged solutions. Setting up SCEP can be hard and tedious. With a plethora of options available, why reinvent the wheel?

This page intentionally left blank

12

IMPLEMENTING **SCEP** ON A **CISCO DEVICE**

Cisco literally invented SCEP, so it's no surprise that devices such as their ASA security appliances support it. However, as with other devices, SCEP is not simple to set up on Cisco hardware.

Before we get into this, I'll throw out a caveat: Stop right now if you are not familiar with the Cisco Internetwork Operating System (IOS, as opposed to iOS), configuring Cisco devices, and SSL. As tricky as it can be to set up SCEP on other platforms, making a mistake with a network appliance can ruin your entire day or year. Make sure you have good backups of your running configuration, and don't commit any changes until you've tested them.

WHY IS **SECURITY** SO **HARD?**

This is something that blows my mind, and I say this as an IT person with two decades of experience. SCEP is, in theory, a great concept. It solves a *lot* of problems, yet setting it up is either impossible or extremely tedious and confusing. Really, your only sane option is to just throw money at the problem. Mind you, it isn't just SCEP. SSL (Secure Sockets Layer), which involves a huge part of what SCEP does, is not any better. It's a mess of what I call "prove your IT worth." It implies that if you cannot handle byzantine interfaces, implementation details surrounded by minefields, and poor or missing documentation, then you shouldn't be allowed to use the technology.

Ironically, the people behind this tech will complain that not enough people are implementing secure networks. Well, no fooling. If drivers had to weave seatbelts by hand and build the frame attach points and the instructions sucked, no one would use them, either. On behalf of IT people and users everywhere, I plead to the people creating this technology: Stop torturing your customers and users. Stop it! It's the 21st century. It is inexcusable that setting up secure networks requires us to play these ego-driven games. If you want secure networks, stop making it so difficult to implement them.

TAKING THE **INITIAL STEPS**

The only real documentation I could find on implementing SCEP on Cisco devices was on the Cisco website. So, yeah, it was not written for "normals" at all. I know that some people are using Cisco devices for SCEP, but getting any real-world information from them proved effectively impossible (not because they didn't want to provide it, but because they weren't sure if they could tell me how to do so without giving away details they didn't want given away). Even according to my Cisco rep, the online documentation I used for this chapter was all they had, and it was based on a device in a clean, lab setup, not the real world. So, here's another plea to anyone reading this: If you can come up with a reliable, repeatable way to set up SCEP that can be explained to people who aren't Cisco-certified, publish it on a website somewhere. You'll be a hero.

For this chapter, we're assuming that you're using a Cisco ASA 5500 series security appliance running Cisco IOS 8.x. You'll also need the Cisco AnyConnect VPN version 2.4 or later. (The AnyConnect VPN client for iOS is available from the App Store.)

Here's a very high-level overview of how SCEP works with Cisco devices:

1. A device with the AnyConnect client connects to an ASA that has a group policy and an XML connection profile set up for SCEP. The device connection has to fail the initial attempt at certificate authentication (because if it doesn't, it doesn't need the cert, and the whole SCEP process never happens). So, the device should not have a valid connection certificate installed on it. Also, if someone successfully logs in to the correct group associated with the policy et al, enrollment will be automatic.

2. The device sends a request for a certificate with parameters that are defined in the XML profile attached to the policy.

3. The CA on the device automatically denies or approves the request.

4. The device downloads the certificate via SCEP.

In this scenario, you're using the ASA only for the certificate setup. You'll want to configure a different web server to perform the actual profile install.

THE **ANYCONNECT SCEP** SETTINGS

The AnyConnect VPN uses XML-based settings profiles to configure the client. To configure SCEP, you'll need to add a few specific elements to the connection profile:

- <AutomaticSCEPHost>: This is the fully-qualified DNS name of the ASA or the IP address of the ASA. This also includes the name of the connection profile/tunnel group that is configured for SCEP enrollment, for example:

  ```
  <AutomaticSCEPHost> asa.bynkii.com/ios_scep
  →</AutomaticSCEPHost>
  ```

- <CAURL>: This identifies the SCEP CA server and contains the challenge password setting, along with the thumbprint (MD5 or SHA1 hash) of the CA cert:

  ```
  <CAURL PromptForChallengePW="true" Thumbprint="245F2342D14D223
  →45245A1234523C2234567">http://ca.bynkii.com</CAURL>
  ```

- <CertificateSCEP>: This defines how the contents of the certificate are requested. This block contains several elements, as in this example:

  ```
  <CertificateSCEP>
      <CADomain>bynkii.com</CADomain>
      <Name_CN>%USER%</Name_CN>
      <Department_OU>Curmudgeonry</Department_OU>
      <Company_O>Misanthropic Yoyodyne</Company_O>
      <State_ST>FL</State_ST>
      <Country_C>US</Country_C>
      <Email_EA>%USER%@bynkii.com</Email_EA>
      <Domain_DC>bynkii.com</Domain_DC>
      <DisplayGetCertButton>false</DisplayGetCertButton>
  </CertificateSCEP>
  ```

The elements within the block are pretty simple to dope out. CADomain is the domain of the certificate authority (CA), in this example, bynkii.com. Name_CN, or Common Name, is the user name. By using the %USER% variable, the username used when logging into the ASA is automatically applied here.

Department_OU is the user's department. Company_O is the company name, State_ST is the state, Country_C is the country. Email_EA is the user's email address, and again we use %USER% substitution to simplify things for the user. Domain_DC is the domain component, which is also bynkii.com in this case. Since we don't want the manual Get Certificate button displayed, we set that to false.

If we put this all together in a client profile, the SCEP section would look like:

```
<CertificateEnrollment>
    <AutomaticSCEPHost>asa.bynkii.com/ios_scep</AutomaticSCEPHost>
    <CAURL PromptForChallengePW="false" Thumbprint="245F2342D14D2234
    →5245A1234523C2234567" >
        http://ca.bynkii.com
    </CAURL>
    <CertificateSCEP>
        <Department_OU>Curmudgeonry</Department_OU>
        <Company_O>Misanthropic Yoyodyne</Company_O>
        <State_ST>FL</State_ST>
        <Country_C>US</Country_C>
        <Email_EA>%USER%@bynkii.com</Email_EA>
        <Domain_DC>bynkii.com</Domain_DC>
        <DisplayGetCertButton>false</DisplayGetCertButton>
    </CertificateSCEP>
</CertificateEnrollment>
```

The full profile is fairly huge, and not particularly germane to this chapter. If you need that level of help with your ASA, I highly recommend contacting your Cisco rep or a Cisco consultant.

CONFIGURING THE **ASA**

So far, we have our XML file set. Now the task is to get it onto the ASA.

First, you need to have the connection group and the policy that the XML profile will be attached to. You'll then want to create an alias on your ASA that points to that group, and put that alias in the `<AutomaticSCEPHost>` section of the XML file.

Configure the connection group so that the XML profile you're going to upload to the ASA is attached to that group.

Finally, set up appropriate access controls for the group.

Then go to the SSL VPN Client settings and upload the XML profile to the device. Next, add a certificate enrollment group policy that uses the SSL VPN Client (such as AnyConnect) as the tunneling protocol. Enable and configure Split Tunneling. In the SSL VPN Client settings, use the XML profile you uploaded as the file in the "Client Profile to Download" settings.

With that done, you'll need to create certificate authentication settings. Create another group policy for this with the appropriate settings for your setup. (I'm sorry to be so general here, but authentication settings are almost infinite in number, and ASAs offer *many* ways to handle authentication.)

Next, create a certificate enrollment connection profile. In this profile, make sure that the alias matches the value in the `<AutomaticSCEPHost>` section of the profile. Also, be sure to match the group policy for this connection profile with the certificate enrollment group policy that you created earlier.

Then, create a certificate authentication connection profile. Make sure the alias and group policy names match the name you set up for the certificate authentication group policy.

Now enable the `<AutomaticSCEPHost>` alias by going into the AnyConnect Connection Profiles settings on the ASA, and enabling the following login page setting:

> Allow user to select connection profile, identified by its alias, on the login page. Otherwise, DefaultWebVPNGroup is the connection profile.

(Yes. That's the name of the setting. Can you tell Cisco is only good at talking to Cisco people?)

TESTING IT ALL

With any luck, all you have left to do is fire up the AnyConnect client on the iOS device, and connect to your ASA. Point the client at the certificate enrollment profile you created, enter the correct username and password, tap Connect, tap Enroll, and bang, done.

If it doesn't work (and there are numerous places you can go sideways or become completely inverted in this process), call a good consultant or your Cisco rep.

The documentation I used for this chapter is available at www.cisco.com/en/US/products/ps6120/products_configuration_example09186a0080b25dc1.shtml and at www.cisco.com/en/US/docs/security/vpn_client/anyconnect/anyconnect24/administration/guide/ac03features.html.

WRAPPING UP

Be aware that Cisco has gotten better about supporting multiple platforms in recent years, but they still really assume that everyone uses Windows. However, that's what makes this so much fun. (Want even more fun? The OS on the ASA is called IOS, which is not iOS, the OS on the iPhone/iPad/iPod Touch. With that in mind, search Cisco's support documentation for "iOS." Wheee! You get a hit for everything! Wheeee! Are we having fun yet?)

By now, I think we've had enough of SCEP to last us . . . well, a very long time. Next up, we'll move on to the very, very cool thing that is Mobile Device Management, and managing iOS devices the way we really, really want to.

This page intentionally left blank

PART III

MOBILE DEVICE **MANAGEMENT**

13

PERFORMING MOBILE DEVICE **MANAGEMENT**

Mobile Device Management? Isn't that what this book has been doing up to now? Managing mobile devices? Well, yes and no. Mobile Device Management in the context of this chapter is a specific concept that addresses some of the liabilities of "manual" configuration profile use in a way that gives you greater device control and makes device management nicer for your users.

THE PROBLEM WITH CONFIGURATION PROFILES

Configuration profiles, as we've used them so far, are handy, but they have some weaknesses. First, we can't just push them out to devices. Even with the "convenience" of SCEP, the device user still has to go to the URL, log in, enter a password, and so on. It's kind of a pain. Profile updates exacerbate that pain.

Configuration profiles, as we've used them, are also rather monolithic, which make changes inconvenient. As an example, let's say that you have to change the SSID of your wireless network. This requires changing only a single line of text in a configuration profile. Yet to change that one line, you have to upload a complete profile and then send the users an email/text message with a URL where they must reinstall that entire profile. Just to change one line of text. Lame.

Configuration profiles also don't allow you to do some tasks remotely. For example, you have decided to take passcodes seriously and implement passcodes with letters, numbers, and special characters. You already know what's going to happen once you implement this change. You're going to get a call from a user, on someone else's phone, because he forgot his passcode. This is a bummer, because you can set passcode policy remotely using profiles, but you can't actually reset or temporarily disable the passcode and allow the user to get into his phone. If this person is at the start of a week-long series of meetings in a remote location, he's not going to be happy with you.

Yes, OTA enrollment is handy, but it's only part of the story, and in many ways, the smallest part. It automates initial enrollment nicely, but then you're still stuck with the problems we just listed—lots of sending of emails and texts on the administrator's part, lots of logging in and tapping on the user's part. The iPhone Configuration Utility (iPCU) and OTA enrollment are an incomplete solution.

Wouldn't it be great if you could push changes out to devices? And just push the changes you actually need? Or push new settings? How about being able to temporarily drop that passcode so the person can get into his phone, yet still require a new passcode within *n* minutes? That would be awesome, wouldn't it?

Yes, yes it would, and thanks to Mobile Device Management, you can do all that and more.

GROKKING THE MOBILE DEVICE MANAGEMENT CONCEPT

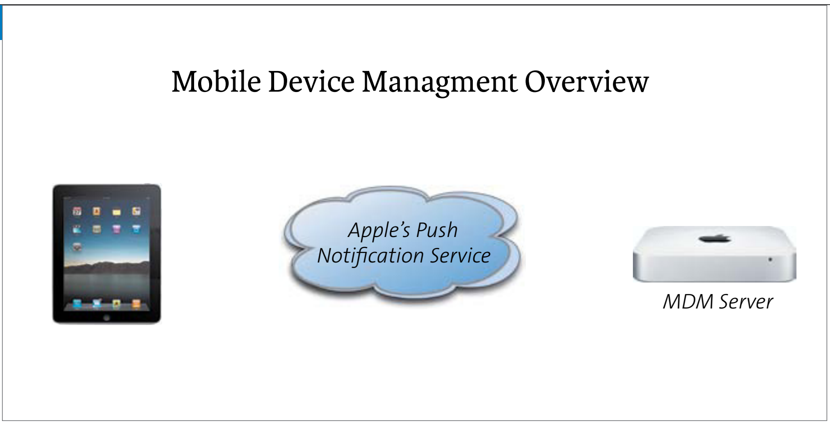
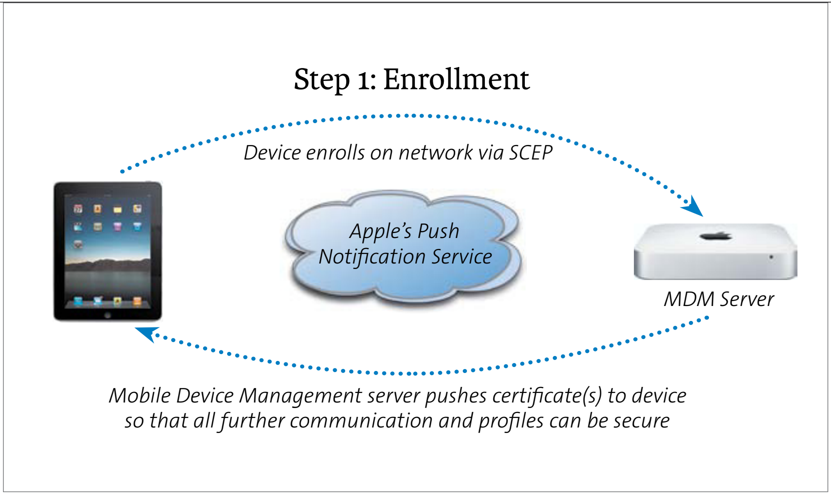

Mobile Device Managment Overview
Apple's Push Notification Service
MDM Server

FIGURE 13.1 Mobile Device Management components

Mobile Device Management is a marriage of push services, configuration profiles, and SCEP. The basic components of Mobile Device Management are the MDM server, Apple Push Services, and the iOS device (**Figure 13.1**). The basic flow of Mobile Device Management is pretty simple.

FIGURE 13.2 Initial enrollment into Mobile Device Management


Step 1: Enrollment
Device enrolls on network via SCEP
Apple's Push Notification Service
MDM Server
Mobile Device Management server pushes certificate(s) to device so that all further communication and profiles can be secure

The device user first enrolls her device to the Mobile Device Management server using SCEP (or some other system) to acquire the necessary certs. This is critical to enable encryption of all further communication with the Mobile Device Management server, configuration profiles, and configuration data (**Figure 13.2**).

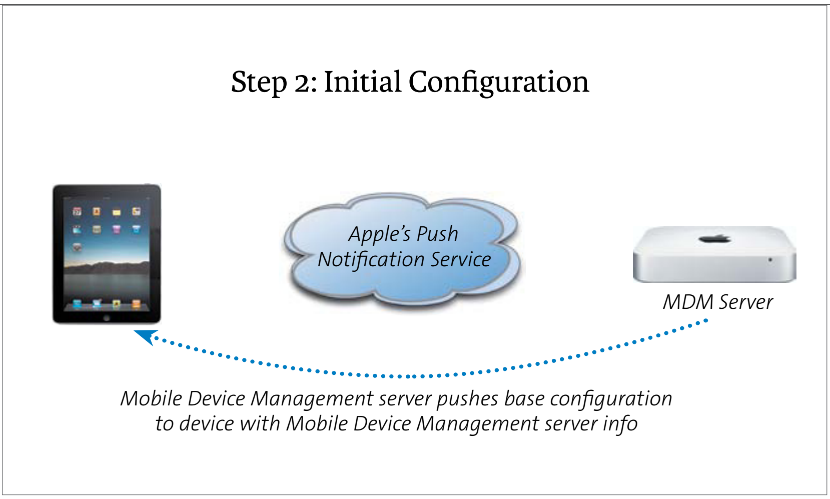

Step 2: Initial Configuration
Apple's Push Notification Service
MDM Server
Mobile Device Management server pushes base configuration to device with Mobile Device Management server info

FIGURE 13.3 Initial Mobile Device Management configuration

After the device is enrolled, a minimal configuration profile with Mobile Device Management information is pushed out to the device (**Figure 13.3**). This profile allows the MDM system to interact with the device without any user involvement.

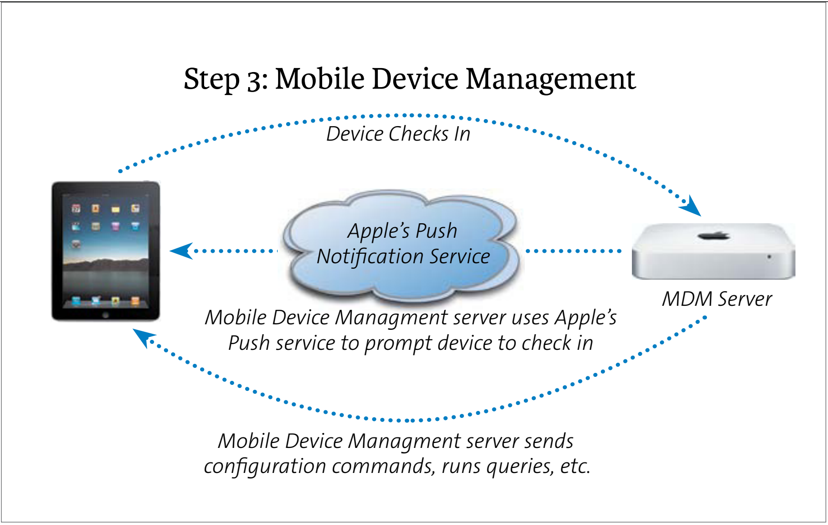

Step 3: Mobile Device Management
Device Checks In
Apple's Push Notification Service
MDM Server
Mobile Device Managment server uses Apple's Push service to prompt device to check in
Mobile Device Managment server sends configuration commands, runs queries, etc.

FIGURE 13.4 Standard Mobile Device Management steps

From this point, the interactions between device and server follow the same process: The server uses Apple Push Notification Service to inform the device that it needs to check in with the server (**Figure 13.4**). The device checks in with the server the next time it is able to do so. The server then sends out configuration changes, runs queries against the device, and so on.

WRAPPING **UP**

The setup for Mobile Device Management isn't significantly harder than setting up conventional OTA delivery of configuration profiles, yet for both the administrator and the user, the advantages are significant.

Mobile Device Management completes your OTA solution by automating all of the process. The user no longer needs to log in for every change or to manually report details. The combination of Apple Push Services and automated OTA services handles this for us.

In addition, it allows us to do some things that are really almost impossible using the manual processes of conventional profile downloads. We said in the example at the start of this chapter, without Mobile Device Management, if a user forgets his passphrase, he's out of luck. His device can't log into the website to download the profile. Using Mobile Device Management, we can silently push out new configurations to the device and resolve this problem.

In the next few chapters, we'll delve into Mobile Device Management details and show why it's something that anyone managing more than a handful of iOS devices should seriously consider.

14

MOBILE DEVICE MANAGEMENT **FEATURES**

In Chapter 13, we talked about some of the advantages that Mobile Device Management has over ”traditional” OTA management and configuration profiles, but we didn’t really get any details of what that means.

FLEXIBILITY AND POWER

Flexibility and power—those are the two words that describe the advantages of Mobile Device Management over the other management methods we've explored so far.

Mobile Device Management easily excels in flexibility. You can change or remove settings without any user involvement.

Mobile Device Management also excels in power because you can apply its flexibility to one or hundreds of devices at once. Note that with Mobile Device Management, *you* apply. Using manual profile installation, even OTA, you notify the users of a new profile, and then wait for them to download and install it. Maybe they will, maybe they won't. If not installing it causes them some pain, well, that's their motivation, but they're still going to be unhappy with you. (After all, it's always *IT's* fault.)

With Mobile Device Management, you don't have to wait for them and they don't have to tap. Devices are configured, inventory is updated under your control, and it all just works. Even better, because the Mobile Device Management implementations out there include all the benefits of SCEP and none of its (*considerable*) pain, this all happens in a secure fashion. Bonus!

But, it's boring to talk about this in the abstract, right? Let's see some examples that illustrate why you would want to use Mobile Device Management.

MANAGING **PASSCODES**

Passphrases, passcodes, and passwords—no matter what word you use for them, they are an alphanumeric pain for every IT person and user on the planet. Unfortunately, they're also a fact of life. We have to deal with them, so we welcome any help we can get. Within Mobile Device Management, we can both remotely set and remove passcodes from devices.

SETTING PASSCODES

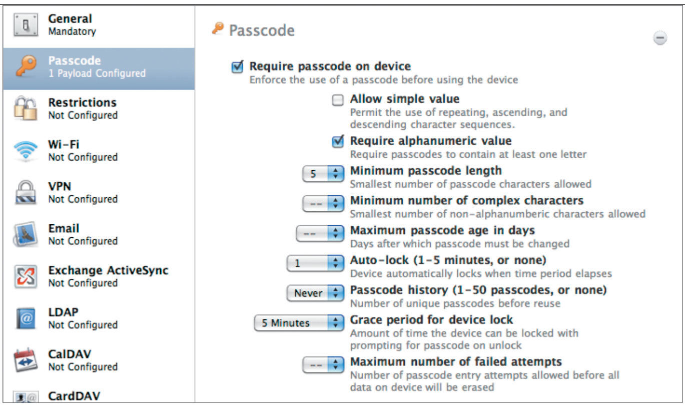
You set passcodes as you would set anything. You create the profile with the correct settings and push it to the device.

ON **PROFILES**

Just in case this hasn't been stated clearly, as this book is long, and I see it quite blearily . . . *all* iOS device configuration settings that you add from an external source are implemented using configuration profiles, whether delivered via USB and the iPhone Configuration Utility, profiles on a web site, or Mobile Device Management servers. As you'll see in this chapter, the advantage to using Mobile Device Management lies in the flexibility you have when managing those profiles.

Let's pretend that we have to set a consistent passcode policy for all of our devices. They cannot be simple (that is no "1234" or similarly lazy passcodes). They must have at least one letter, and a minimum length of five characters.

We also want the device to autolock after one minute, and to be unlockable for five minutes after an autolock without reentering a passcode. Finally, we won't keep a history of passcodes, we won't change the passcode on a regular basis, we won't require special characters such as "&", and we won't be erasing the device if too many wrong passcode entry attempts are made.



General
Mandatory

Passcode
1 Payload Configured

Restrictions
Not Configured

Wi-Fi
Not Configured

VPN
Not Configured

Email
Not Configured

Exchange ActiveSync
Not Configured

LDAP
Not Configured

CalDAV
Not Configured

CardDAV

Passcode

[✓] **Require passcode on device**
Enforce the use of a passcode before using the device

[ ] **Allow simple value**
Permit the use of repeating, ascending, and descending character sequences.

[✓] **Require alphanumeric value**
Require passcodes to contain at least one letter

5 — **Minimum passcode length**
Smallest number of passcode characters allowed

-- — **Minimum number of complex characters**
Smallest number of non-alphanumberic characters allowed

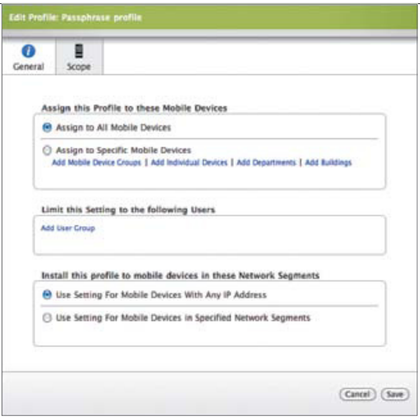
-- — **Maximum passcode age in days**
Days after which passcode must be changed

1 — **Auto-lock (1-5 minutes, or none)**
Device automatically locks when time period elapses

Never — **Passcode history (1-50 passcodes, or none)**
Number of unique passcodes before reuse

5 Minutes — **Grace period for device lock**
Amount of time the device can be locked with prompting for passcode on unlock

-- — **Maximum number of failed attempts**
Number of passcode entry attempts allowed before all data on device will be erased

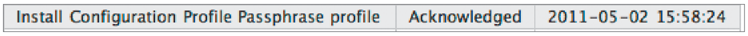


Edit Profile: Passphrase profile

Assign this Profile to these Mobile Devices
(●) Assign to All Mobile Devices
( ) Assign to Specific Mobile Devices
Add Mobile Device Groups | Add Individual Devices | Add Departments | Add Buildings

Limit this Setting to the following Users
Add User Group

Install this profile to mobile devices in these Network Segments
(●) Use Setting For Mobile Devices With Any IP Address
( ) Use Setting For Mobile Devices in Specified Network Segments



Install Configuration Profile Passphrase profile	Acknowledged	2011-05-02 15:58:24

FIGURE 14.1 (top left) Passcode settings configuration

FIGURE 14.2 (right right) Profile scope settings

FIGURE 14.3 (bottom) Profile push results

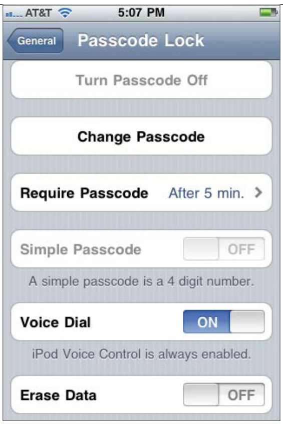
First, we create a profile with the appropriate settings (**Figure 14.1**). Then, we select the devices we want to push the profile to and push it to each device (**Figure 14.2**).

In the case of JAMF's Casper (which appears in these screenshots), that's all we need to do. The settings are then pushed out to the devices. We can verify this in the Management console (**Figure 14.3**).

So what does the user see on his device? Well, starting from a no-passcode state, once the new profile is pushed, the next time the user accesses the device, he'll see an alert dialog telling him to set a passcode within the next 60 minutes.

If the user wishes, he can ignore this alert for the next hour and use his device. He'll get nagged a lot but he can keep ignoring it until he gets to the end of the 60-minute grace period. Then he will have to enter a passcode.

He must tap Continue, enter a passcode that meets the requirements you've set, and confirm the passcode to set it.


AT&T 5:07 PM

General | Passcode Lock

Turn Passcode Off

Change Passcode

Require Passcode: After 5 min.

Simple Passcode: OFF

A simple passcode is a 4 digit number.

Voice Dial: ON

iPod Voice Control is always enabled.

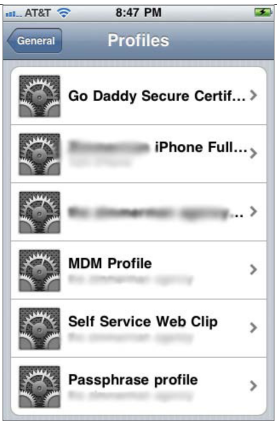
Erase Data: OFF

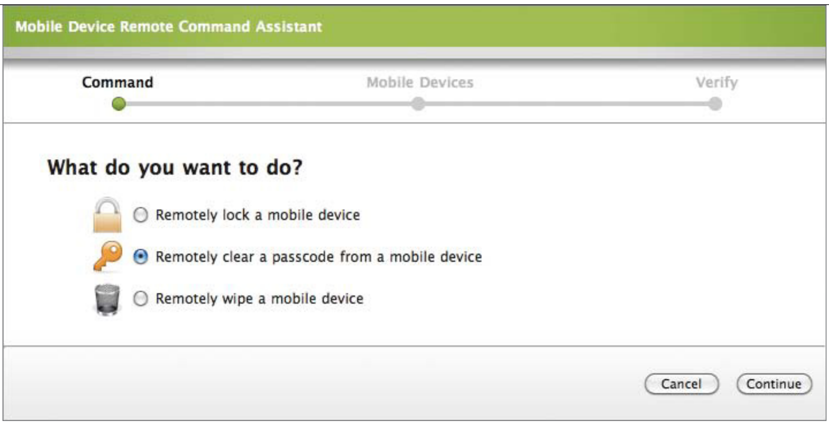
FIGURE 14.4 New passcode settings

If you now look at the passcode settings on the device, you'll see a mixture of user-configurable settings and settings that are locked. Some of the user-configurable settings are manually set, such as the "Require Passcode After *n* min," setting. Don't worry. Apple's not actually letting you bypass things. For example, in **Figure 14.4**, you can see that the profile automatically set the interval to five minutes. The user can change that to lock after five minutes, one minute, or immediately. In other words, the user can opt for more restrictive settings than you entered, but not less. I can live with that.

The same thing is true of the Erase Data setting. Because we didn't set it, the user can opt to turn it on because that choice would be *more* restrictive.

Finally, if we look at the profiles on the device (**Figure 14.5**), we can see the profile we pushed out to the device to start this process.




Mobile Device Remote Command Assistant

Command — Mobile Devices — Verify

What do you want to do?

- ○ Remotely lock a mobile device
- ● Remotely clear a passcode from a mobile device
- ○ Remotely wipe a mobile device

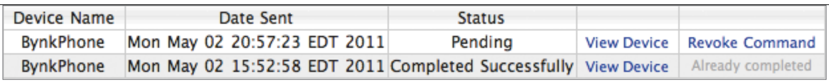
FIGURE 14.5 (left) Profiles showing passcode profile

FIGURE 14.6 (right) Initiate the clear passcode command

This is great. We have pushed a profile to all our devices, they're passcode-protected, and life is good. At least it's good until the phone rings, and that traveling user can't seem to remember a passcode more complicated than "1111." He's forgotten his passcode. Groan. He's also a thousand miles away. You don't know what passcode he set, and of course, his entire world will collapse if he can't get into his devices.

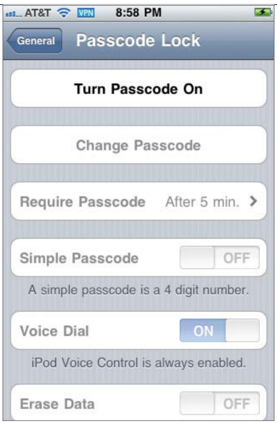
Without Mobile Device Management, this is a major pain. With Mobile Device Management, it's only a minor annoyance because you can easily disable a passcode.

First, you tell your Mobile Device Management server that you want to clear a passcode from a remote device (**Figure 14.6**). You select the device and then confirm



Device Name	Date Sent	Status		
BynkPhone	Mon May 02 20:57:23 EDT 2011	Pending	View Device	Revoke Command
BynkPhone	Mon May 02 15:52:58 EDT 2011	Completed Successfully	View Device	Already completed

FIGURE 14.7 Passcode removal command pending



AT&T VPN 8:58 PM

General | Passcode Lock

Turn Passcode On

Change Passcode

Require Passcode: After 5 min.

Simple Passcode: OFF

A simple passcode is a 4 digit number.

Voice Dial: ON

iPod Voice Control is always enabled.

Erase Data: OFF

FIGURE 14.8 On-device settings showing passcode removed

that you want to clear the passcode from the selected device, and send the command to do so. You'll see that the command is pending (**Figure 14.7**).

Within a few minutes (less than five if the device is reachable), the passcode is cleared. The device will be usable without a passcode, and even the settings will show that the passcode has been disabled (**Figure 14.8**).

Well, that's great, but we still want the device to be passcode protected! Do we now have to resend the profile? Nope. Within a few minutes, the device will once again display the 60-minute nag countdown to set a passcode as it did when you initially pushed out the profile.

Considering the amount of pain that IT people normally experience with such situations, let me tell you, this ease of management is *Awesome*. Also, it's a way to deal with anyone who thinks he's just found a clever way to make his passcode headaches go away.

Keep in mind that part of this solution (the remote removal of the passcode) is impossible without using Mobile Device Management. Further, unless you use Mobile Device Management and its push abilities, you're not going to have an easy time handling mass passcode setup.

Let's review the IT and user steps involved in pushing out a profile, *any* profile, without MDM:

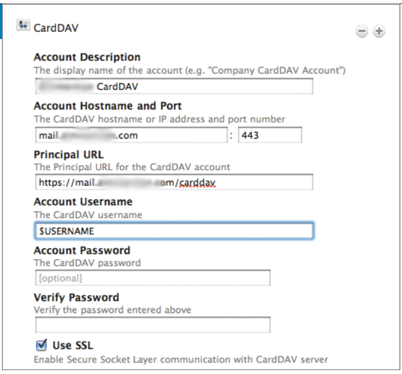
1. Build the profile in the iPhone Configuration Utility and put it on a web server.
2. Send an email or text to users requesting they go to the website in the link and install the new profile. (Send an individual email or text for every device you need to configure/modify.)
3. The user taps a link to go to web site.
4. The user logs into website.
5. The user downloads and installs the profile and performs whatever steps are required by the profile.

Now, same thing, with MDM:

1. Build the profile in the iPhone Configuration Utility or the Mobile Device Management server tool.
2. Push the profile to the devices you select.
3. The user performs minimal configuration (enter passcode, provide password for server, and so on).

Cake! Mobile Device Management makes this process and others a lot easier. Let's look at setting up a CardDAV server.

MANAGING **CARDDAV** SETTINGS



CardDAV

Account Description
The display name of the account (e.g. "Company CardDAV Account")
[redacted] CardDAV

Account Hostname and Port
The CardDAV hostname or IP address and port number
mail.[redacted].com : 443

Principal URL
The Principal URL for the CardDAV account
https://mail.[redacted]om/carddav

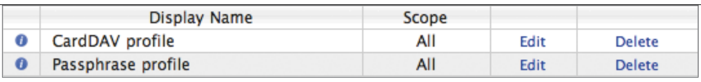
Account Username
The CardDAV username
$USERNAME

Account Password
The CardDAV password
[optional]

Verify Password
Verify the password entered above

☑ **Use SSL**
Enable Secure Socket Layer communication with CardDAV server

FIGURE 14.9 CardDAV settings in the profile



	Display Name	Scope		
●	CardDAV profile	All	Edit	Delete
●	Passphrase profile	All	Edit	Delete

FIGURE 14.10 Profile with delete/edit options

INSTALLING THE CARDDAV PROFILE

The procedure here is virtually identical to installing the passcode profile. First, set up your profile with the CardDAV info (**Figure 14.9**). Notice the $USERNAME variable entered in the Account Username field. This is one of those "why I pay for things" features. When the profile is pushed to the device, the username associated with that device is automatically filled in. It's a minor thing, but it's really nice to have; and it means that the device user only has to enter her password for the CalDAV server.

Save the profile, assign it to the correct devices, and within seconds, those devices, if reachable, will have the new profile. When I say "seconds," I'm speaking literally. It took me longer to navigate from the Casper settings to the device settings than it did to push the profile. A minute or so of work, a few seconds to a few minutes of network traffic, a password tapped on a device, and (bang!) your users have CardDAV. Or email, or CalDAV, or what have you.

REMOVING THE CARDDAV PROFILE

Removing the profile is even simpler. Open the profile you want to remove (**Figure 14.10**), click Delete, or click Edit if you're removing a profile setting but keeping the profile.

The profile is removed, along with the CardDAV settings, and the device's user has performed zero work. This is how these things are supposed to work: easily, quickly, and securely with a minimum of manual fiddling about for everyone.

All this is nice, but what about when you want to gather statistics on devices? Mobile Device Management comes to the rescue again!

GATHERING DEVICE **INVENTORY/INFORMATION**

Even if you are locking devices down, some information is going to be tedious to get without using Mobile Device Management. Device versions, device names, IMEI numbers, Wi-Fi MAC addresses, software versions, available space, model, currently-installed apps, installed certs, configuration profiles, and on and on. Some of these, such as the Wi-Fi MAC address, won't change much. But others will, and if you can't easily gather that information remotely, you have to manually collect it, or have the user try to read it to you. Yeah, that's *big* fun.

Wouldn't it be great if you could just collect this information in one place, and review it as necessary? That would be awful handy when you needed to know how many of your devices could support the latest iOS update, wouldn't it? What you want is the kind of information a Mobile Device Management server can give you.

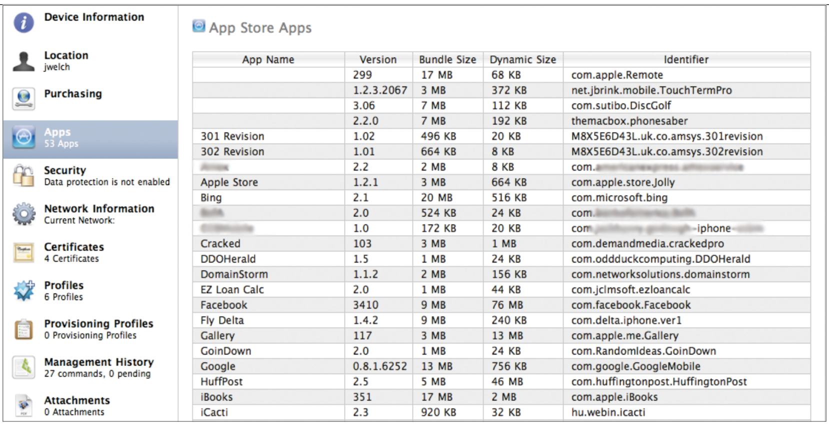
Need details on a device? How about determining which apps are installed on a device? (**Figure 14.11**) What about locating certificates? (**Figure 14.12**) Need to know the configuration profiles installed on a device? (**Figure 14.13**)

Done, done, done, and done. That is the flexibility and power you get with Mobile Device Management, and that is why it is such an improvement over past management methods.

WRAPPING **UP**

With Mobile Device Management, you can actually *manage* your devices in a proactive way. (Yes, I know that according to pundits, you can't do this with iOS devices. Oh look! The pundits are wrong. *Again*!)

In Chapter 15, we'll look at what's involved in setting up a Mobile Device Management server using Casper from JAMF software.



Device Information

Location
jwelch

Purchasing

Apps
53 Apps

Security
Data protection is not enabled

Network Information
Current Network:

Certificates
4 Certificates

Profiles
6 Profiles

Provisioning Profiles
0 Provisioning Profiles

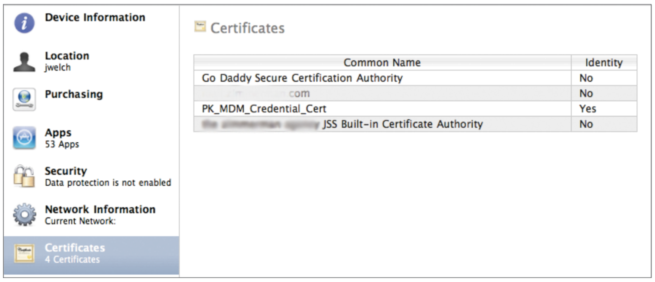
Management History
27 commands, 0 pending

Attachments
0 Attachments

App Store Apps

App Name	Version	Bundle Size	Dynamic Size	Identifier
	299	17 MB	68 KB	com.apple.Remote
	1.2.3.2067	3 MB	372 KB	net.jbrink.mobile.TouchTermPro
	3.06	7 MB	112 KB	com.sutibo.DiscGolf
	2.2.0	7 MB	192 KB	themacbox.phonesaber
301 Revision	1.02	496 KB	20 KB	M8X5E6D43L.uk.co.amsys.301revision
302 Revision	1.01	664 KB	8 KB	M8X5E6D43L.uk.co.amsys.302revision
[redacted]	2.2	2 MB	8 KB	com.[redacted]
Apple Store	1.2.1	3 MB	664 KB	com.apple.store.Jolly
Bing	2.1	20 MB	516 KB	com.microsoft.bing
[redacted]	2.0	524 KB	24 KB	com.[redacted]
[redacted]	1.0	172 KB	20 KB	com.[redacted]-iphone-[redacted]
Cracked	103	3 MB	1 MB	com.demandmedia.crackedpro
DDOHerald	1.5	1 MB	24 KB	com.oddduckcomputing.DDOHerald
DomainStorm	1.1.2	2 MB	156 KB	com.networksolutions.domainstorm
EZ Loan Calc	2.0	1 MB	44 KB	com.jclmsoft.ezloancalc
Facebook	3410	9 MB	76 MB	com.facebook.Facebook
Fly Delta	1.4.2	9 MB	240 KB	com.delta.iphone.ver1
Gallery	117	3 MB	13 MB	com.apple.me.Gallery
GoinDown	2.0	1 MB	24 KB	com.RandomIdeas.GoinDown
Google	0.8.1.6252	13 MB	756 KB	com.google.GoogleMobile
HuffPost	2.5	5 MB	46 MB	com.huffingtonpost.HuffingtonPost
iBooks	351	17 MB	2 MB	com.apple.iBooks
iCacti	2.3	920 KB	32 KB	hu.webin.icacti

FIGURE 14.11 App inventory



Device Information

Location
jwelch

Purchasing

Apps
53 Apps

Security
Data protection is not enabled

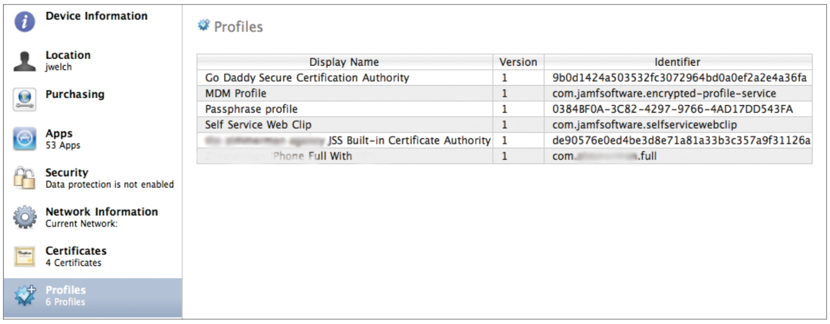
Network Information
Current Network:

Certificates
4 Certificates

Certificates

Common Name	Identity
Go Daddy Secure Certification Authority	No
[redacted] com	No
PK_MDM_Credential_Cert	Yes
[redacted] JSS Built-in Certificate Authority	No

FIGURE 14.12 Certificate inventory



Device Information

Location
jwelch

Purchasing

Apps
53 Apps

Security
Data protection is not enabled

Network Information
Current Network:

Certificates
4 Certificates

Profiles
6 Profiles

Profiles

Display Name	Version	Identifier
Go Daddy Secure Certification Authority	1	9b0d1424a503532fc3072964bd0a0ef2a2e4a36fa
MDM Profile	1	com.jamfsoftware.encrypted-profile-service
Passphrase profile	1	0384BF0A-3C82-4297-9766-4AD17DD543FA
Self Service Web Clip	1	com.jamfsoftware.selfservicewebclip
[redacted] JSS Built-in Certificate Authority	1	de90576e0ed4be3d8e71a81a33b3c357a9f31126a
[redacted] Phone Full With	1	com.[redacted].full

FIGURE 14.13 Configuration profile inventory

15

SETTING UP A MOBILE DEVICE MANAGEMENT **SERVER**

In this chapter, I talk about setting up a Mobile Device Management server. Because of the differences between doing this for Mac OS X 10.6 Server and Mac OS X Server 10.7, I'll split this chapter along those lines. For Mac OS X 10.6 Server and earlier, I'll be using Casper. For Mac OS X Server 10.7, I'll use Apple's Profile Manager.

But first let's talk a bit about what you need to think about and plan before you start downloading and installing things.

DO YOU REALLY NEED TO **RUN** YOUR OWN **SERVER?**

Although I've mostly talked about Casper in previous chapters, the truth is, you can use quite a few cloud-based Mobile Device Management servers in the software as a service (SAAS) model. Given the way Mobile Device Management works, there's a certain amount of logic behind using the cloud, especially if you don't have the facilities or expertise to devote to managing iOS devices and running the server underneath the Mobile Device Management package.

If you just want to manage your iOS devices, but don't want to run another server, or add this onto an existing server, then a cloud service is an excellent option. Just *do* pay attention to things like SLAs and response times. Especially what constitutes a response. Remember "Hi, this is <person> from <provider>, we see you have a problem, a technician will be contacting you in four to six business days," delivered within four hours at 3 am on a Sunday does in fact meet a 24x7x4 response time requirement. It also sucks, but contracts are about the letter of the law, not the spirit, so do make sure the letter says what you think it should.

HOW **BIG** SHOULD YOUR **SERVER** BE?

As it turns out, capacity planning for Mobile Device Management servers is still a bit of an unrefined art. For everything other than applications, you're dealing with small bursts of data. Configuration profiles are on the low end of the kilobyte scale, so almost any fairly modern network and Internet connection is up to the bandwidth task. Yes, apps can get into megabyte sizes, but even so, we're talking low megabytes and it's unlikely that you're pushing apps on a daily basis.

Also, thanks to Mobile Device Management, you don't have to supervise the push. If it takes a day or so to fully update a few hundred devices, that's something you can plan for and manage. You probably don't need a huge dedicated server farm for this task or even a dedicated server at all. While testing for this book, my hardware was a Mac Mini server with 8 GB of RAM with two 500 GB hard drives. Admittedly, my device pool size is *small*, but my needs didn't even make the Mini breathe hard.

So, if all you're doing with the Mobile Device Management server is Mobile Device Management stuff, you don't need to plan for a $30K hardware buy, although your situation may make a big bucks server desirable. (I say this because some of the MDM products, such as Casper or Absolute Manage, do a *lot* more than Mobile Device Management, and if you're going to use them to manage your desktop computers, for example, you will need a bigger server.) However, for most companies in the SMB market, a server with the same rating as Mac Mini (or two) should be able to manage a lot of devices.

FIREWALL PLANNING

Because of the way Mobile Device Management works, you need two kinds of connectivity. Obviously, you'll need a connection between your server and your devices.

In addition, you'll need to talk to certain Apple services such as the Apple Push Notification Server (gateway.push.apple.com), the Apple site for determining if an app is too big for your cellular provider's network and must be downloaded via Wi-Fi (ax.init.itunes.apple.com), and the Apple site for validating distribution certificates (ocsp.apple.com).

To support access to those sites, some nonstandard, iOS-specific ports must be opened:

- Port 2195, used by your Mobile Device Management server to talk to the Apple Push Notification Server
- Port 5223, used by the Apple Push Notification Server to talk to iOS devices

Further, you may need to open port 443 and/or port 80 through your Mobile Device Management server's firewall. If you're using Casper, you may also need to allow port 8443, Casper's default SSL port.

If opening any of those ports is an issue in your system, you may want to seriously consider using a cloud-based service.

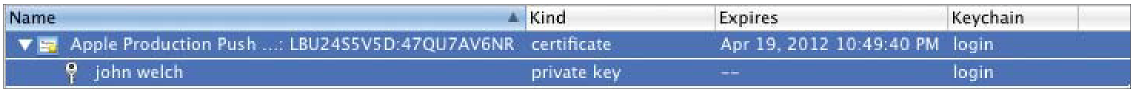
GETTING A **PUSH NOTIFICATION** CERTIFICATE

To use Mobile Device Management, you *must* have a push notification certificate from Apple. To get that certificate, you *must* be enrolled in the iOS Enterprise Developer Program (http://developer.apple.com/programs/ios/enterprise/), which as of this writing costs $299/year.

When you're a part of the program, getting the certificate is cake in the "Portal" sense of the word. (The cake, in this case, is not a lie, but you will do a lot of jumping through hoops to get it.) JAMF has an excellent how-to document that applies to Mac OS X, Windows, and Linux. You can read it at www.jamfsoftware.com/libraries/pdf/white_papers/JAMFSoftware-Creating_an_Apple_Push_Notification_Certificate.pdf. The basic steps for getting the certificate are:

1. On your Mobile Device Management server, generate an SSL CSR.
2. Go to the Apple iOS Dev Center, and enter the iOS Provisioning Portal.
3. In the App IDs section, create a new App ID. Enter a description and a bundle identifier. This *must* start with `com.apple.mgmt` and end with a unique string, for example, `com.apple.mgmt.com.bynkii.com`.
4. In the list of certificates you'll get, look for the one with the bundle ID you just entered, and click Configure.
5. Select the "Enable for Apple Push Notification" checkbox, and click the Configure button for "Production Push SSL Certificate."
6. The wizard that appears allows you to upload the CSR you created and generate the push cert. Do what the nice wizard tells you to do.
7. Save the cert to your computer.

NOTE: The preceding steps specifically apply to Mac OS X Server. If you're installing Casper on Windows or Red Hat Linux, your world will be a bit different, and I recommend you follow the documentation at www.jamfsoftware.com/libraries/pdf/white_papers/Installing_the_JAMF_Software_Server_on_Alternate_Platforms.pdf.



Name	Kind	Expires	Keychain
Apple Production Push ...: LBU24S5V5D:47QU7AV6NR	certificate	Apr 19, 2012 10:49:40 PM	login
john welch	private key	--	login

FIGURE 15.1 Push cert and private key

Since you're going to be using this cert with Casper in this chapter, you need to do a few more things.

1. Install the cert you downloaded into the keychain by double-clicking the cert and installing it into the system keychain or your login keychain. (Or really, any keychain you can access because this step is just a starting point to exporting the cert to a different format. I used my login keychain because doing so sometimes makes it easier to get to the private key for the cert.)

   Note that if you don't have the Apple WWDR intermediate cert installed, Keychain Access will gripe about your cert being signed by an unknown authority. If you go to Certificates in the iOS Provisioning Portal, you can download that cert from there and install it into the same keychain that you installed the push cert. This doesn't really affect anything, but it avoids that annoying red text.

2. Select both the Push cert *and* the private key (**Figure 15.1**) and export *both* of them as a .p12 bundle. You'll be asked to create a password to secure the bundle. That's a good idea. Make sure you remember this password because you'll need it later.

3. Save the .p12 bundle somewhere, and then quit Keychain Access.

USING **MAC OS X SERVER 10.7**

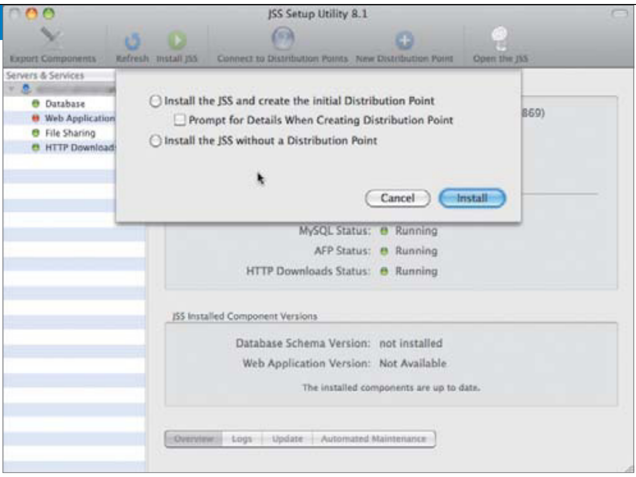
Setting up push management with Mac OS X Server 10.7 is easier than in earlier versions. Open the server application, and log into the server running Profile Manager. Select the server in the "Hardware" section, then click the Settings tab. On the line that reads "Enable Apple push notifications", click the Edit button. Then, enter an Apple ID that has an email address attached to it. (The ID itself doesn't have to be an email address, it just has to have one associated with it. If you don't have an Apple ID, there's a button there for you to create one.)

At this point, you just enter the required info and click Acquire Certificates. Assuming your Apple ID is valid, the whole process is completed in about a minute.

Once the certs are acquired, click OK, and select the checkbox to enable push notifications, and you're done. Simple as can be.

As it turns out, I already went over this in Chapter 10, "Implementing SCEP on Mac OS X Server." So rather than repeat everything here, I'll refer you back to that chapter, and the section "Implementing SCEP on Mac OS X Server 10.7."

INSTALLING **CASPER** ON **MAC OS X 10.6 SERVER**



◯ Install the JSS and create the initial Distribution Point
☐ Prompt for Details When Creating Distribution Point
◯ Install the JSS without a Distribution Point

MySQL Status: ● Running
AFP Status: ● Running
HTTP Downloads Status: ● Running

JSS Installed Component Versions

Database Schema Version: not installed
Web Application Version: Not Available

The installed components are up to date.

FIGURE 15.2 Distribution point options

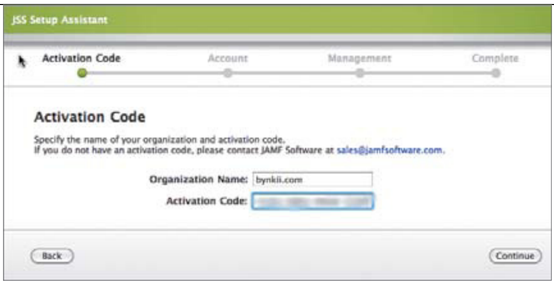
So, we've got our server set up, and we have our copy of Casper. We've configured our firewall correctly and have a push cert, so we're pretty much good to go.

One of the nicer aspects of Casper is that you don't have to install and configure it from the server itself. Casper uses SSH for the installation and configuration steps. So, if you don't have easy physical access to the server and don't want to do everything via Apple Remote Desktop or VNC, you don't have to.

1. Mount the Casper disk image, and after reading the documentation (because you *always* read the documentation before you start, right?), start the JAMF Software Server (JSS).

2. Give it the DNS name/IP address of the server you're installing Casper on, along with a username and password that have both SSH and administrator privileges. The JSS setup will think a minute, and then offer you two options: "Install the JSS with a Distribution Point" and "Install the JSS without a Distribution Point" (**Figure 15.2**).

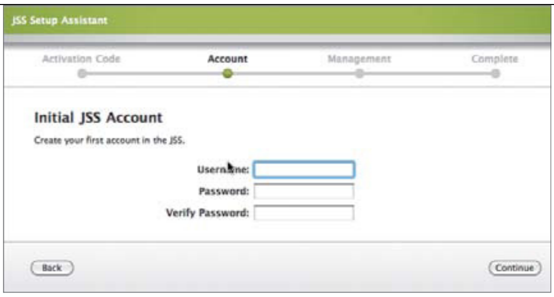
NOTE: We are setting up Casper as a Mobile Device Management server only. It can actually do quite a bit more, but we can't cover the entire breadth of Casper's services in this book. However, if you are looking for a framework for managing your Macs and your iOS devices, Casper is an excellent option.



Activation Code

Specify the name of your organization and activation code.
If you do not have an activation code, please contact JAMF Software at sales@jamfsoftware.com.

Organization Name: bynkii.com
Activation Code:



Initial JSS Account

Create your first account in the JSS.

Username:
Password:
Verify Password:

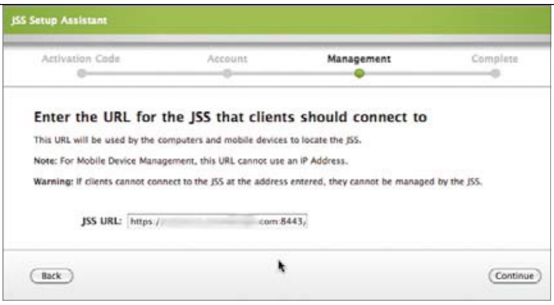
FIGURE 15.3 Activation code

FIGURE 15.4 Initial JSS administrator account

3. Because we're only interested in Mobile Device Management, choose "Install the JSS without a Distribution Point." The JSS Setup Utility will finish the initial install and open a web browser page that allows you to finish the setup and manage Casper.

4. Enter your Organization/Company Name and the Activation Code for Casper (**Figure 15.3**), followed by the initial administrator account and password (**Figure 15.4**).

NOTE: Just in case you haven't picked up on it, Casper and JSS are interchangeable for our purposes. Technically, JSS is a superset that includes Casper.



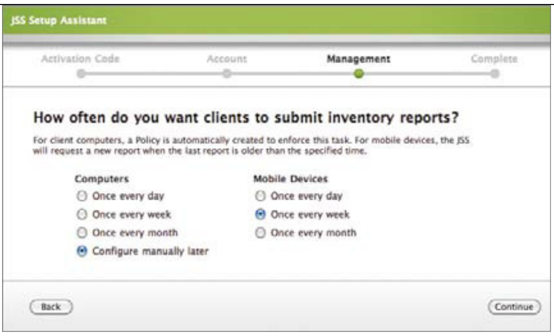
Enter the URL for the JSS that clients should connect to

This URL will be used by the computers and mobile devices to locate the JSS.

Note: For Mobile Device Management, this URL cannot use an IP Address.

Warning: If clients cannot connect to the JSS at the address entered, they cannot be managed by the JSS.

JSS URL: https://[redacted].com:8443/



How often do you want clients to submit inventory reports?

For client computers, a Policy is automatically created to enforce this task. For mobile devices, the JSS will request a new report when the last report is older than the specified time.

Computers
- ○ Once every day
- ○ Once every week
- ○ Once every month
- [✓ Configure manually later]

Mobile Devices
- ○ Once every day
- [✓ Once every week]
- ○ Once every month

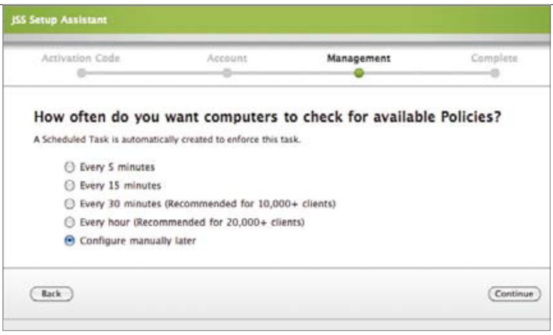
FIGURE 15.5 JSS URL

FIGURE 15.6 Inventory report settings

5. Enter the URL that clients will use to communicate with Casper (**Figure 15.5**). If you don't have a reason to use a different URL, use the default URL provided.

6. Enter the inventory report frequency (**Figure 15.6**). I would really like it if Casper was better able to be set up as *just* a Mobile Device Management server or computer management server, and let us pick "never" as an option. But, since it doesn't, and we are performing Mobile Device Management only, set the Computers option to "Configure manually later" and set the Mobile Devices option to whatever will work best for you.

NOTE: Given the Mac OS X Server reliance on DNS, this tip may be redundant, but just in case: You really, really, really want to make sure that the DNS setup for this server is correct. This DNS name will be used by devices when they enroll and the URL that you'll use to manage the server. Incorrect DNS here will cause a never-ending stream of problems.

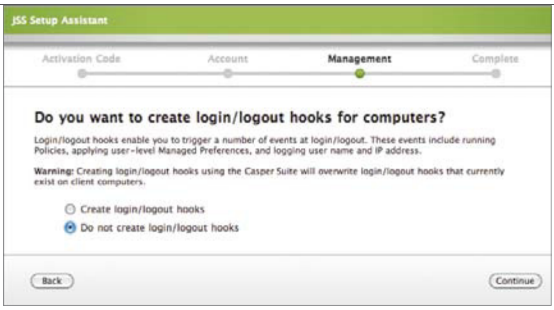


How often do you want computers to check for available Policies?

A Scheduled Task is automatically created to enforce this task.

- Every 5 minutes
- Every 15 minutes
- Every 30 minutes (Recommended for 10,000+ clients)
- Every hour (Recommended for 20,000+ clients)
- [✓ Configure manually later]

FIGURE 15.7 Policy check settings



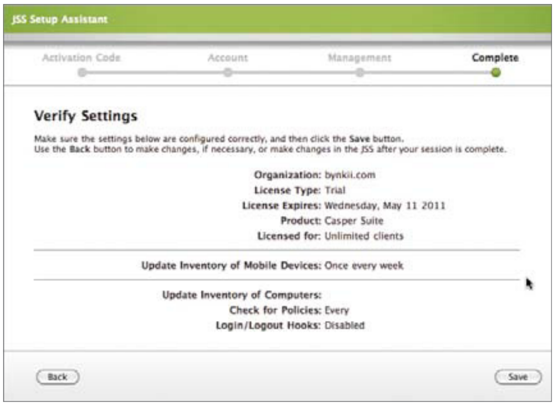
Do you want to create login/logout hooks for computers?

Login/logout hooks enable you to trigger a number of events at login/logout. These events include running Policies, applying user-level Managed Preferences, and logging user name and IP address.

Warning: Creating login/logout hooks using the Casper Suite will overwrite login/logout hooks that currently exist on client computers.

- Create login/logout hooks
- [✓ Do not create login/logout hooks]

FIGURE 15.8 Login/logout hook settings



Verify Settings

Make sure the settings below are configured correctly, and then click the **Save** button.
Use the **Back** button to make changes, if necessary, or make changes in the JSS after your session is complete.

Organization: bynkii.com
License Type: Trial
License Expires: Wednesday, May 11 2011
Product: Casper Suite
Licensed for: Unlimited clients

Update Inventory of Mobile Devices: Once every week

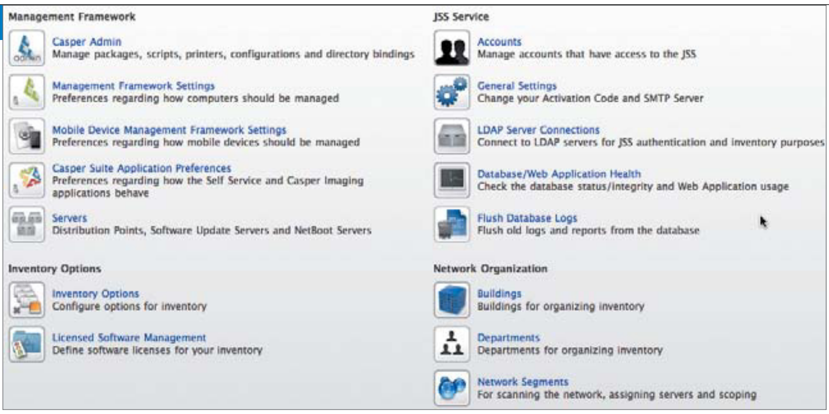
Update Inventory of Computers:
Check for Policies: Every
Login/Logout Hooks: Disabled

FIGURE 15.9 Final confirmation and save

7. You'll have to decide how often you want computers to check for available policies (**Figure 15.7**). Since we don't care in this context, set this to "Configure manually later." The next option will ask if you want to create login/logout hooks for computers (**Figure 15.8**). Again, we don't care here, so set this to "Do not create login/logout hooks."

8. Verify your settings (**Figure 15.9**) and click Save.

CONFIGURING **CASPER** FOR MOBILE DEVICE **MANAGEMENT**



Management Framework

Casper Admin
Manage packages, scripts, printers, configurations and directory bindings

Management Framework Settings
Preferences regarding how computers should be managed

Mobile Device Management Framework Settings
Preferences regarding how mobile devices should be managed

Casper Suite Application Preferences
Preferences regarding how the Self Service and Casper Imaging applications behave

Servers
Distribution Points, Software Update Servers and NetBoot Servers

Inventory Options

Inventory Options
Configure options for inventory

Licensed Software Management
Define software licenses for your inventory

JSS Service

Accounts
Manage accounts that have access to the JSS

General Settings
Change your Activation Code and SMTP Server

LDAP Server Connections
Connect to LDAP servers for JSS authentication and inventory purposes

Database/Web Application Health
Check the database status/integrity and Web Application usage

Flush Database Logs
Flush old logs and reports from the database

Network Organization

Buildings
Buildings for organizing inventory

Departments
Departments for organizing inventory

Network Segments
For scanning the network, assigning servers and scoping

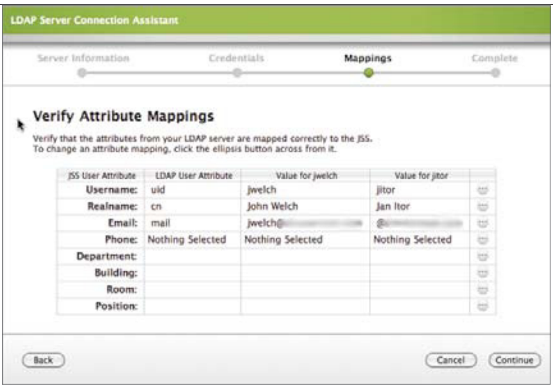
FIGURE 15.10 Casper settings page

With the Casper installation done, you'll see a page with a list of links that you can use to set up various features of Casper. You can run these from this page, or within Casper itself using the Settings page (**Figure 15.10**).

CONFIGURING LDAP

If you aren't using LDAP, (that is, Open Directory, Active Directory, or OpenLDAP) to run your server, you can skip this section.

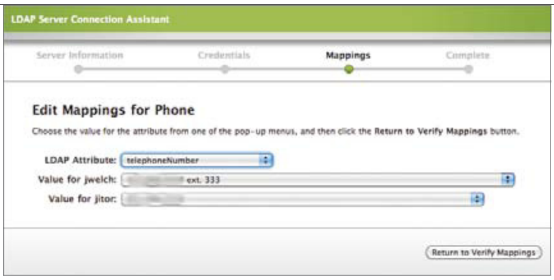
I highly recommend that you connect Casper to LDAP because it makes some Mobile Device Management tasks *much* easier, such as automatically entering email addresses and user names. Click LDAP Server Connections, and then click the Add LDAP Server Connection button.



Verify Attribute Mappings

Verify that the attributes from your LDAP server are mapped correctly to the JSS.
To change an attribute mapping, click the ellipsis button across from it.

JSS User Attribute	LDAP User Attribute	Value for jwelch	Value for jitor
Username:	uid	jwelch	jitor
Realname:	cn	John Welch	Jan Itor
Email:	mail	jwelch@[redacted]	@[redacted]
Phone:	Nothing Selected	Nothing Selected	Nothing Selected
Department:			
Building:			
Room:			
Position:			



Edit Mappings for Phone

Choose the value for the attribute from one of the pop-up menus, and then click the **Return to Verify Mappings** button.

LDAP Attribute: telephoneNumber
Value for jwelch: [redacted] ext. 333
Value for jitor: [redacted]

FIGURE 15.11 Attribute mappings settings

FIGURE 15.12 Sample attribute mapping

In the wizard that pops up, choose your directory type, and then just do what the wizard asks. The wizard is straightforward for the most part. Only the attribute mappings section can be a bit confusing (**Figure 15.11**).

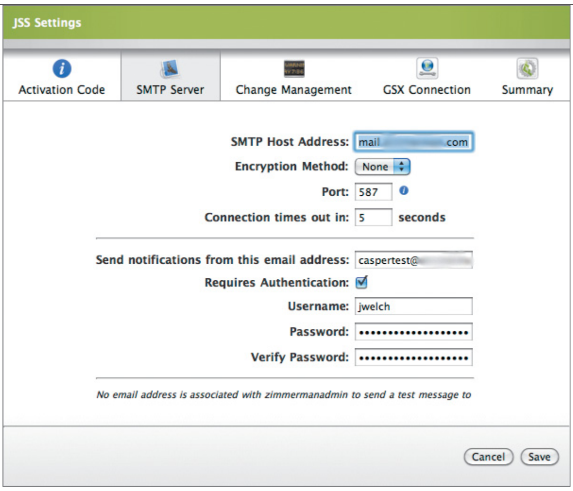
The mappings don't much apply to iOS devices, but if you want to correctly integrate Casper into your LDAP infrastructure, you should set up the mappings by telling Casper the attributes your LDAP server uses for phone, department, and so on. You want to do this because not every implementation of LDAP uses the same attributes for the same data. To set the mappings, click the ellipsis in the row you want to configure, and select the applicable LDAP attribute (**Figure 15.12**).

You'll also be asked to enter two LDAP user group names, which are used to verify group membership for the users you used to configure the LDAP mappings. Once that's done, click Save, and Casper can now talk to your LDAP server.

LDAP MAPPINGS **WARNING**

If you have no idea what LDAP attribute mappings are, I *highly* recommend you leave this setup to someone who understands it, or set it aside until you understand it. Although Casper won't actually cause harm if you get the mappings wrong, you can create some serious confusion for yourself with incorrect mappings.

FIGURE 15.13 SMTP settings



SMTP Host Address: mail[redacted].com
Encryption Method: None
Port: 587
Connection times out in: 5 seconds

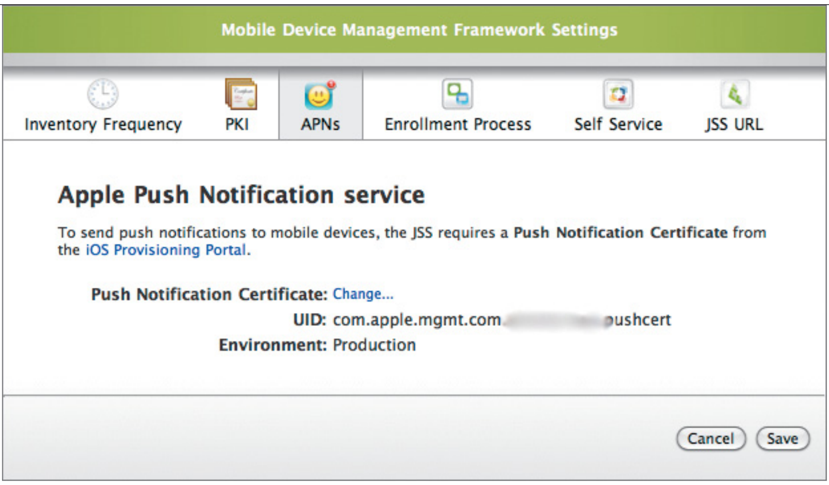
Send notifications from this email address: caspertest@[redacted]
Requires Authentication: ✓
Username: jwelch
Password: ••••••••••••••••••••
Verify Password: ••••••••••••••••••••

No email address is associated with zimmermanadmin to send a test message to

CONFIGURING EMAIL SETTINGS

Since you probably want Casper to email you with various notifications, you'll want to set up an SMTP connection. If you don't care about email notifications, feel free to skip this section.

1. To set up an SMTP connection for Casper, click General Settings on the main Casper settings page, and then click SMTP Server.
2. Fill in the settings (**Figure 15.13**).
3. Click Save. (Kudos to JAMF for handling authenticated SMTP well; it's not that common.)



Apple Push Notification service

To send push notifications to mobile devices, the JSS requires a **Push Notification Certificate** from the iOS Provisioning Portal.

Push Notification Certificate: Change...
UID: com.apple.mgmt.com.[redacted]pushcert
Environment: Production

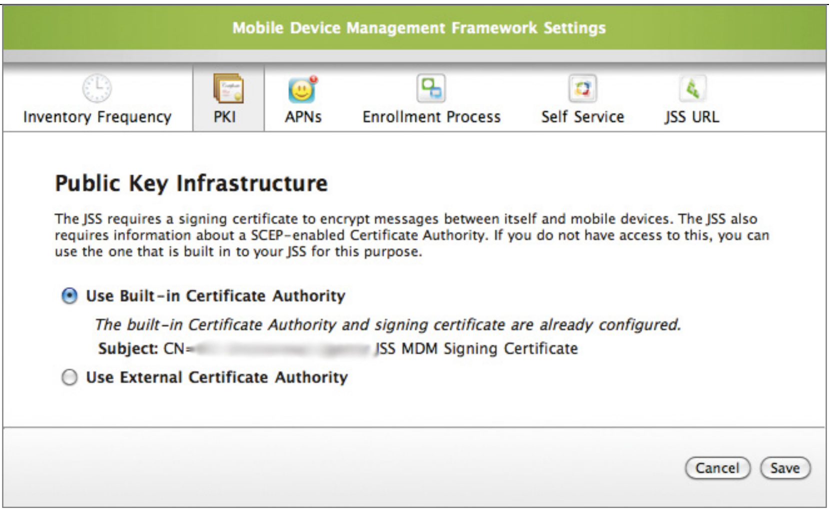
FIGURE 15.14 Completed Push Notification Certificate install

UPLOADING THE PUSH NOTIFICATION CERTIFICATE

To use the Mobile Device Management functionality of Casper (or any Mobile Device Management server), you have to install the Push Notification Certificate you previously downloaded and exported.

1. From the main settings page, click Mobile Device Management Framework Settings and then click the APNs tab (**Figure 15.14**).
2. Click the Upload link, and upload the .p12 cert you exported.
3. You'll also need to enter the password that you created for the .p12 cert when you exported it.
4. Click Save, and you're done.

FIGURE 15.15 Completed root CA setup



Public Key Infrastructure

The JSS requires a signing certificate to encrypt messages between itself and mobile devices. The JSS also requires information about a SCEP-enabled Certificate Authority. If you do not have access to this, you can use the one that is built in to your JSS for this purpose.

[✓ Use Built-in Certificate Authority]

The built-in Certificate Authority and signing certificate are already configured.

Subject: CN=[redacted] JSS MDM Signing Certificate

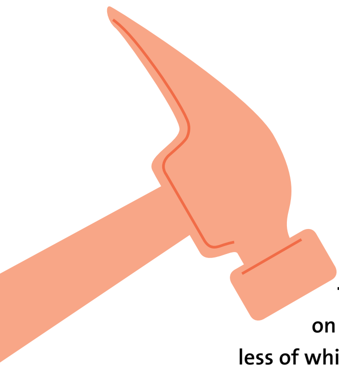
Use External Certificate Authority

SETTING UP THE SCEP SERVER

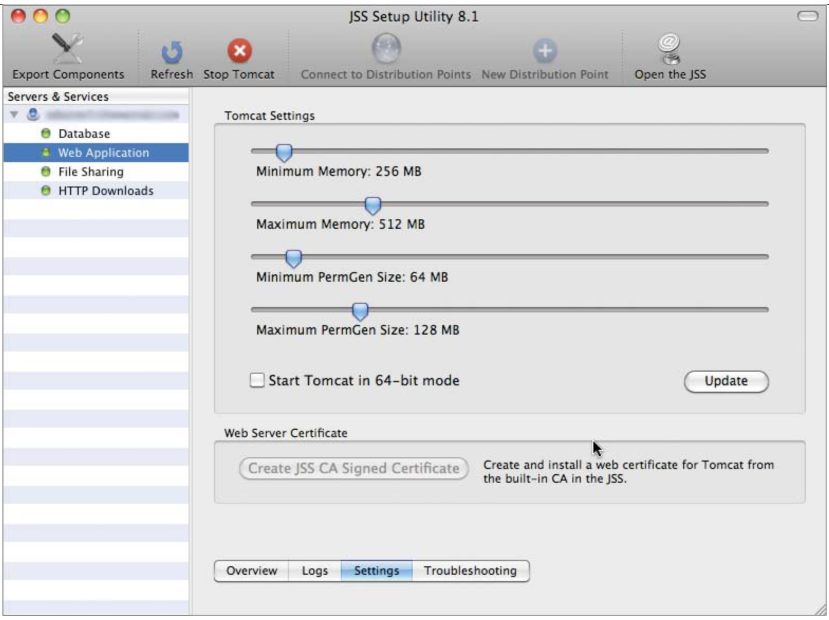
Casper comes with built-in SCEP functionality, or you can use an external SCEP server. For this section, let's use the built-in functionality.

Along with SCEP, Casper allows you to use your own certificate authentication (CA) or use its built-in CA feature. Again, to keep things simple for this section, we'll use the built-in CA and other certificate functionality.

1. From the main settings page, click Mobile Device Management Framework Settings.
2. Click the PKI tab, and then select Use Built-In Certificate Authority (**Figure 15.15**).
3. For the subject, type something like *CN=My Company's JSS Mobile Device Management Signing Certificate.*
4. Select a key size (which should be at *least* 2048), and then enter a keystore password.
5. Click Save.



TIP: Considering how painful SCEP can be to set up on its own, I recommend setting up this option regardless of which Mobile Device Management product you use.



Minimum Memory: 256 MB

Maximum Memory: 512 MB

Minimum PermGen Size: 64 MB

Maximum PermGen Size: 128 MB

Start Tomcat in 64-bit mode

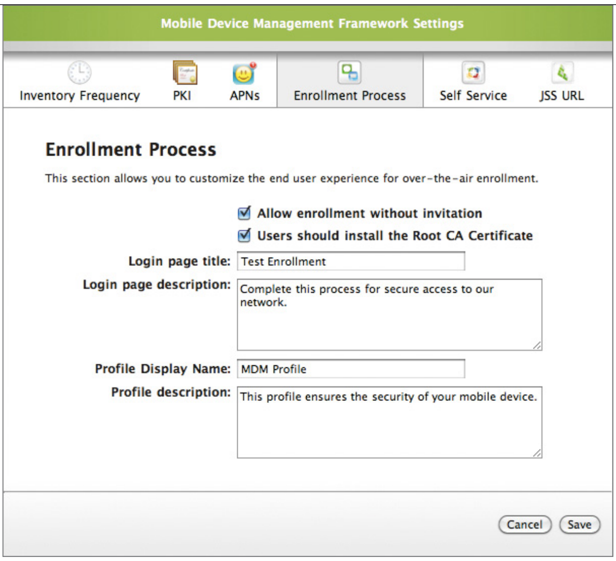
Create and install a web certificate for Tomcat from the built-in CA in the JSS.

FIGURE 15.16 Web Server cert setup in the JSS Setup Utility

You'll also need to create a Web server cert to secure communications between the iOS device and the Casper server. One thing to keep an eye on is that if you have any kind of cert already installed for the web server, even the default cert that Mac OS X Server creates for you, you can't have Casper also set up a cert.

In that situation, you're better off deleting the default installed cert, setting up Casper, and then creating a new cert if you need one. To set up a web server cert with Casper:

1. Start up the JSS Setup Utility, connect to the Casper server, and select Web Application (**Figure 15.16**).
2. Click the Create JSS CA Signed Certificate button to create the cert.
3. When the setup utility is done, click OK.



Enrollment Process

This section allows you to customize the end user experience for over-the-air enrollment.

[✓] Allow enrollment without invitation

[✓] Users should install the Root CA Certificate

Login page title: Test Enrollment

Login page description: Complete this process for secure access to our network.

Profile Display Name: MDM Profile

Profile description: This profile ensures the security of your mobile device.

FIGURE 15.17 Initial MDM enrollment profile setup

SETTING UP THE INITIAL ENROLLMENT PROFILE

This is the initial MDM profile that is installed on iOS devices when they enroll. To set this up, from the main settings page, click Mobile Device Management Framework Settings, and then click "Enrollment Process" (**Figure 15.17**).

There's not really much you can set up here. You can allow users to enroll without being invited (there are as many reasons to allow this as not), and have the users install the Root CA certificate (a good idea.) You can give the login page the device will see a title and description, and do the same for the profile.

At this point, Casper is basically set up for Mobile Device Management functionality, and you need only set up your configuration profiles and invite devices to enroll.

WRAPPING **UP**

Casper comes with solid documentation, the kind that I dearly wish more companies would spend time on creating. You can download the main Casper product documentation from the JAMF site at www.jamfsoftware.com/support/documentation. You can download additional PDFs, including the all important "Generating an Apple Push Notification Certificate," at www.jamfsoftware.com/resources/pdf-library. I highly recommend perusing those sites, even if you aren't going to use Casper. There's a lot of information that's useful for any iOS administrator regardless of the management package in use.

16

LIMITATIONS OF MOBILE DEVICE **MANAGEMENT**

It would be pretty remarkable if Mobile Device Management (MDM) didn't have any limitations or weaknesses. It does. But for the most part, there are no surprises among them. In fact, only two limitations are worth talking about.

UNDERSTANDING INFRASTRUCTURE **COMPLEXITY**

Of all the ways you can manage iOS devices, Mobile Device Management is the most complex. It's also the most powerful, but it's definitely not simple. It requires that you have an iOS Enterprise Developer program membership, a server, multiple levels of certificates, and the freedom to punch some holes in your firewall. (If you have to deal with network security guardians, this will make them *very* interested in your MDM server.)

Mobile Device Management offers LDAP integration, which is really handy; but again, if you have staffers who manage your directory as their primary jobs, *they* will get very interested in your MDM server.

If you choose a cloud-based MDM solution, you have to manage that cloud relationship, which probably eats up the time you save by not directly managing a server and firewall. If you want LDAP integration using a cloud-based solution, depending on the vendor, *that* integration may be of sufficient complexity to override any cloud-based convenience gains.

If you use MDM in an SAAS implementation, you'll be paying for your MDM service forever, and that fee can increase as you add devices. As a result, you may have the accounting and legal departments *very* interested in your MDM server.

You also have to pay a lot more attention to profile management than you would when using a simpler management setup. Do you have one profile per setting, or do you just modify one uber-profile? If you have many iOS devices, MDM can get rather complex as you group devices with differing policies for each group.

I've often observed that when someone gets a tool that suddenly gives him a *lot* more capability, he can get a little crazy. Do yourself a favor, avoid getting crazy with MDM or you may make your life harder than it was before MDM.

Finally, because of the nature of MDM, your ability to customize it is limited by what your choice of MDM server allows. For some folks, that's okay, they just want it to work, and they don't care if they can twiddle every possible bit. For others, that's a huge issue.

LOCKING MOBILE DEVICE MANAGEMENT **PROFILES**

Another potential issue with MDM is that, by design, you can't lock the root profile the way you can lock a "normal" configuration profile—that is, a configuration profile you created in the iPhone Configuration Utility, and distributed via USB tethering, email, or simple web server download. When using those kinds of configuration profiles, you can effectively prevent the user from removing it without wiping the device.

However, when using MDM, and this is also by design, you can't lock down MDM profiles in the same way. The user can always delete the root MDM profile, and in doing so, delete any other configuration profiles built on top of the root profile. This is an MDM design decision made by Apple. If you want it changed, you'll have to take it up with them.

You can of course, use "traditional" configuration profiles along with MDM, so that some settings can't be changed, but those profiles would have to be installed using non-MDM methods.

WHADDYA MEAN I CAN'T **LOCK THEM?**

I've tried to find some way to look at Apple's decision so that even though I don't agree with it, I can *understand* it. Unfortunately, I can't. I really don't see why you would lock down a configuration profile delivered via one set of methods, including OTA, so that you have to wipe the device to get rid of it. Yet, that very same file, when delivered by Mobile Device Management, cannot be locked down in the same way. You can lock down any individual profile you send out, but if you remove the root Mobile Device Management profile, the other profiles get yanked too, even if you require a passphrase to remove them. They're both configuration profiles, yet you can lock one and not the other. It'd be like throwing away packages delivered by FedEx while keeping anything delivered by UPS to the end of time. It doesn't make sense on any level I can think about it, yet there it is.

WRAPPING UP

I'm not trying to scare you away from Mobile Device Management. I do think it's a fantastic solution to managing iOS devices. But no solution, none, comes without its attendant issues. In the long run, or even in the short run, you are better off knowing about potential issues even if they don't currently apply to you. Even if you never have a flood, it *is* nice to know that you're in a flood zone.

This page intentionally left blank

PART IV

BASIC **WIRELESS APPLICATION** DISTRIBUTION

17

BASIC WIRELESS APPLICATION DISTRIBUTION **BACKGROUND** AND **SETUP**

In previous chapters, we've talked about various methods of configuring devices. However, there's a large part of the iOS universe beyond configuration: apps, including in-house enterprise applications and those that are available on the App Store. So in this and the next few chapters, we'll explore wireless application distribution: what's involved, what's needed, and some strategies, both inside and outside of Mobile Device Management.

BACKGROUND AND REQUIREMENTS FOR **WIRELESS APP DISTRIBUTION**

The idea seems simple enough: You have *n* iOS devices and *m* apps, and you want to install those apps onto the device. It's a simple process via USB, but that solution doesn't scale well. Using USB and the iPhone Configuration Utility (iPCU) gets old quick if you have more than a handful of devices.

Clearly, you want to be able to distribute those apps wirelessly; it's the only way that makes sense.

WIRELESS BY ANY OTHER **NAME**

It is important to remember that iOS devices have two flavors of wireless: Wi-Fi and cellular. If you're using small apps (in terms of the file size, not functionality), you don't have to worry. If you're distributing larger apps, you may not be able to distribute them over cellular networks. For example, the Apple App Store in the United States has a 20 MB cap for app downloads. If you want to acquire an app larger than 20 MB, you have to use Wi-Fi. This is probably not a huge issue in most situations, but it is something to remember if you start to get complaints from users unable to download a large app via a cell network.

First, you'll need to be a member of the iOS Developer Enterprise Program. To do so, you'll need to have a DUNS, or Dun & Bradstreet Number. (For information about DUNS, see http://en.wikipedia.org/wiki/Data_Universal_Numbering_System.) An Enterprise membership will cost you about $300 a year.

Keep in mind that we're talking about creating and distributing in-house, aka enterprise, apps, not App Store apps. Because you can't easily buy those, you'll need to have iOS developers on staff or hire them to program your apps. (That seems obvious, but I'm old enough to remember when people were astonished that you needed a computer to use either the Internet or Windows 95. So, I try to assume that the obvious isn't always obvious.)

Once that's done, your developers (if they aren't you) will have to go to the iOS Portal and create the various distribution certificates and provisioning profiles you'll need. You may also have to register your iOS devices with Apple in the provisioning portal; you can perform a bulk upload of devices via a tab-delimited file. Remember that when you provision an in-house application, you provision it to the device. For that to happen, the device may have to be registered. (Chapter 4 includes more details about this.)

I **HEARD** I HAVE TO **REGISTER?!?**

The whole "register your device with Apple" thing sounds bad to some; but in my experience, it's not a big deal, and it does help prevent J. Random. Malware App from invading your devices. This is a real-world problem that's already happened multiple times in Google's Android app store.

However, the registration requirement applies only to "development" devices, or for people who are not a part of the Enterprise iOS program. If you are distributing "finished" apps (whatever they are) and you are a member of the Enterprise iOS program, then you can distribute "finished" apps to an unlimited number of devices without registering them with Apple. The devices just need the proper provisioning profile installed and you're set.

Keep in mind that within the auspices of internal apps, what you choose to call "development" and "finished" are up to you. Apple offers no hard and fast requirement on this, and doesn't really seem to care. If you want to fling betas at your users, that's entirely up to you. I don't recommend it, but you can. (Yes, I agree, this entire thing is a bit confusing. Frankly, Apple doesn't do a great job of explaining any of it.)

Obviously, you'll need to have actual apps and their associated provisioning profiles (or a company profile for all your devices and apps) so that you have something to distribute. And finally, you'll need a means to distribute them.

WHAT ABOUT **WIRED?**

I'm not going into detail about wired app distribution because it's covered with regard to the iPhone Configuration Utility (iPCU). This chapter, along with the next few chapters, are *technically* about *wireless* app distribution. Also, there's really not much to setting up wired distribution that's different from wireless, except you use either iTunes or the iPCU to push the app to a device. The entire process for iTunes, for example, is described in literally three sentences and a phrase in four numbered list items in the "Distributing Enterprise Apps for iOS 4 Devices" doc on the Apple developer site at http://developer.apple.com/library/ios/#featuredarticles/FA_Wireless_Enterprise_App_Distribution/Introduction/Introduction.html.

Now that you know the requirements and methods of wireless app distribution, why don't we set up a simple distribution server? Fortunately, it's not that hard to do.

APP DISTRIBUTION **SERVER REQUIREMENTS**

First, obviously, you're going to need a web server, and preferably one that is secure. I'm of the opinion that if your server will store data you even *vaguely* care about, and you don't want the server turned into a malware zombie, take the extra time to make it secure. At the very least, use SSL and a userid/password combo. On the app side, thanks to the way Apple manages in-house app distribution, it's really hard to hack an app to install on a device that you haven't registered.

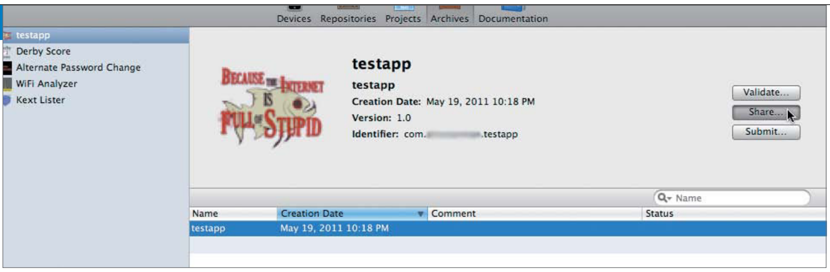
WHAT ABOUT **JAILBROKEN iOS APP** DISTRIBUTION?

I'm not trying to be *overly* snarky here, but for enterprise or in-house app distribution, I see no need or advantage to jailbreaking an iOS device. You aren't going through the Apple App Store, *none* of their rules apply to you. You could write an app that uses only private APIs and breaks every rule in the App Store, but it wouldn't mean anything because Apple would never see it. These are *your* apps distributed to *your* devices. Therefore, a fundamental advantage to jailbreaking, being able to install whatever you want on your devices, doesn't apply to Enterprise Apps.

So, I don't see the point of jailbreaking as it applies to iOS devices in the enterprise. Given the threat of malware and other problems from random applications, jailbreaking isn't a good choice for any IT person worth her salt. If that's the route you want to go, I recommend searching Google or Bing for more information.

Obviously, all your iOS Enterprise Developer setup must be done, including acquiring distribution certificates, provisioning profiles, and so on. You'll also need (just as obviously) an app in an .ipa format and an XML manifest (which I'll get into in just a moment). Finally, you need to ensure that any device using this server to download and install apps can also get to http://ax.init.itunes.apple.com and http://ocsp.apple.com. The first site handles file size limitation if the device is connecting to the server via cellular network, and the second verifies the validity of the distribution certificate and the provisioning profile.

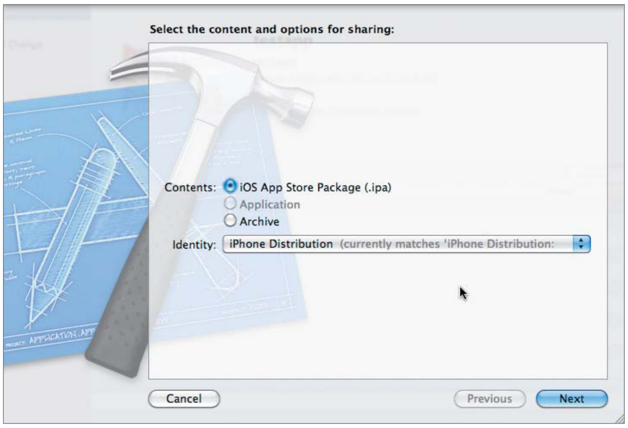
PREPARING THE APP



testapp
testapp
Creation Date: May 19, 2011 10:18 PM
Version: 1.0
Identifier: com.[redacted].testapp

Name	Creation Date	Comment	Status
testapp	May 19, 2011 10:18 PM		

FIGURE 17.1 Xcode Organizer



Select the content and options for sharing:

Contents: [✓ iOS App Store Package (.ipa)] Application Archive

Identity: iPhone Distribution (currently matches 'iPhone Distribution:

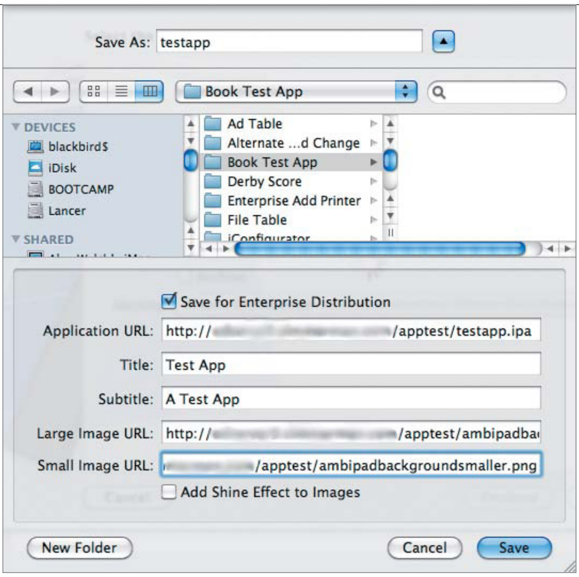
FIGURE 17.2 Sharing options

There's not much to explain here, but let's go through the process anyway. So, I created an iPad app that does nothing. Well, it has text. (Seriously, what I know about Objective C, other than how to spell it, is that it's based on C and has the word "Objective" in it.) But, we don't care about what the app does, just how to set it up and distribute it. So we'll use our do-nothing app.

1. Archive the app so that it appears in the Organizer. (This is all in Xcode 4, by the way.)

2. Click the Share button (**Figure17.1**).

3. Make sure that Contents is set to iOS App Store Package (.ipa), and that your iPhone Distribution identity is enabled (**Figure 17.2**).



☑ Save for Enterprise Distribution

Application URL: http:// /apptest/testapp.ipa

Title: Test App

Subtitle: A Test App

Large Image URL: http:// /apptest/ambipadba

Small Image URL: /apptest/ambipadbackgroundsmaller.png

☐ Add Shine Effect to Images

FIGURE 17.3 Enterprise distribution save options

4. Make sure you enable the "Save for Enterprise Distribution" checkbox, and then click Next and save your app. This opens a new part of the save dialog (**Figure 17.3**) with fields for the application URL (the path to the .ipa file), the title and subtitle for the app URL, a link to a large (512 × 512 pixel) app image, and a link to a small (57 × 57 pixel) app image. There's also an option to add a shine effect to the images.

5. Click Save. When you're done saving, you'll see that two files were created: the app's .ipa file, and a corresponding .plist file. The .plist file is just an XML file that contains the parameters you entered when you saved the app in the Xcode Organizer.

6. Copy the app and the .plist file to the desired location on your web server. See the following sample for an idea of what one .plist file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
→"http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">
```

```
<dict>
    <key>items</key>
    <array>
        <dict>
            <key>assets</key>
            <array>
                <dict>
                    <key>kind</key>
                    <string>software-package</string>
                    <key>url</key>
                    <string>https://aserver.com/apptest/testapp.ipa
                    →</string>
                </dict>
                <dict>
                    <key>kind</key>
                    <string>full-size-image</string>
                    <key>needs-shine</key>
                    <false/>
                    <key>url</key>
                    <string>https://aserver.com/apptest/
                    →ambipadbackgroundlarge.png</string>
                </dict>
                <dict>
                    <key>kind</key>
                    <string>display-image</string>
                    <key>needs-shine</key>
                    <false/>
                    <key>url</key>
                    <string>https://aserver.com/apptest/
                    →ambipadbackgroundsmaller.png</string>
                </dict>
            </array>
            <key>metadata</key>
            <dict>
                <key>bundle-identifier</key>
                <string>com.bynkii.testapp</string>
                <key>bundle-version</key>
                <string>1.0</string>
                <key>kind</key>
                <string>software</string>
                <key>subtitle</key>
                <string>A Test App</string>
                <key>title</key>
                <string>Test App</string>
            </dict>
        </dict>
    </array>
</dict>
</plist>
```

There's really not much to the .plist. You've got the dict that describes the .ipa as a software package and passes the URL to the .ipa file. You've got the URLs to the two image files with the shine option settings. And finally, you've got some application metadata-like bundle-identifier, version, kind, and so on.

After you've saved the .ipa and the .plist files out of the Xcode Organizer and copied them to the desired location on your web server, you're ready to build the page that will have the links needed to install the app on your iOS device.

ACCESSING THE APP DISTRIBUTION **WEB PAGE**

You really need only one thing here: the app URL. All the rest of your effort depends on how much work you want to do to make it look pretty.

The URL itself is a single line:

```
<a href="itms-services://?action=download-manifest&url=
→http://odserver3.zimmerman.com/apptest/testapp.plist">
→Install Test App</a>
```

Notice that this URL doesn't link to the .ipa file, but to the .plist file. That's not a mistake, that's how this works. Remember that the .plist file not only has the URLs for the .ipa file, but also for the image files you'll need. Also, even though the href points to "itms-services," the iTunes store isn't involved.

That's really all you need to allow a device to download this app: the one link and the files, of course. Once those are in place, people just navigate to the page and tap the URL to install the app.

INSTALLING THE APP

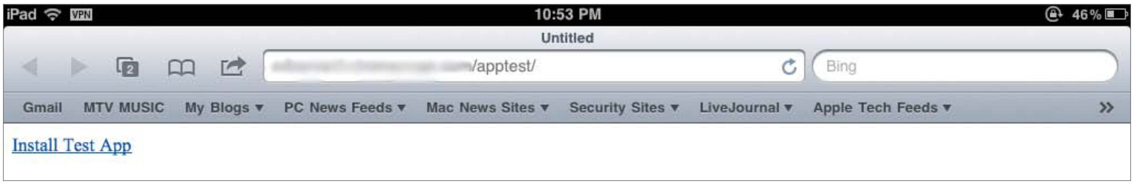


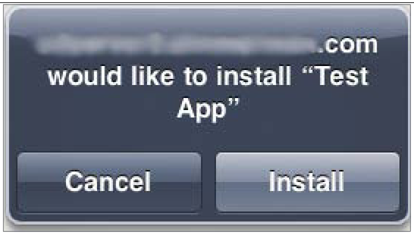
FIGURE 17.4 Simple page with app installation link

As I said, the web page can be as pretty or as stark as your needs and environment dictate. All that *has* to be there is the correct link (**Figure 17.4**).

1. When you tap the link, a dialog appears stating that <server> would like to install <App name> (**Figure 17.5**).
2. Tap Install to start the download and installation. You'll be taken to the device's home screen, where you'll see the customary loading icon (**Figure 17.6**).
3. When the installation is completed, you'll see a standard iOS app icon, ready to be tapped and opened (**Figure 17.7**).
4. The first time you run the app, a dialog appears (**Figure 17.8**) asking if you really want to do that (something I appreciate, even though given the process, it's probably unnecessary).
5. Tap Continue to start the app, and you're off and running.

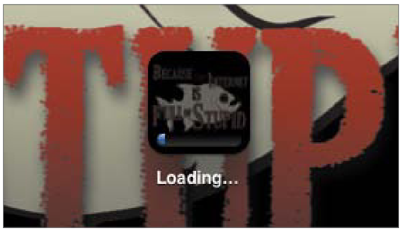
WRAPPING UP

See? There's really not much to the infrastructure here. Honestly, most of the work is in Xcode, and the server part of it is dead simple. In the next chapter, we'll look at doing this using Mobile Device Management.



[redacted].com would like to install "Test App"

FIGURE 17.5 Installation request dialog



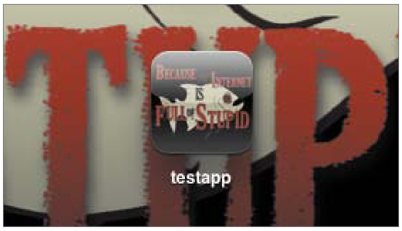
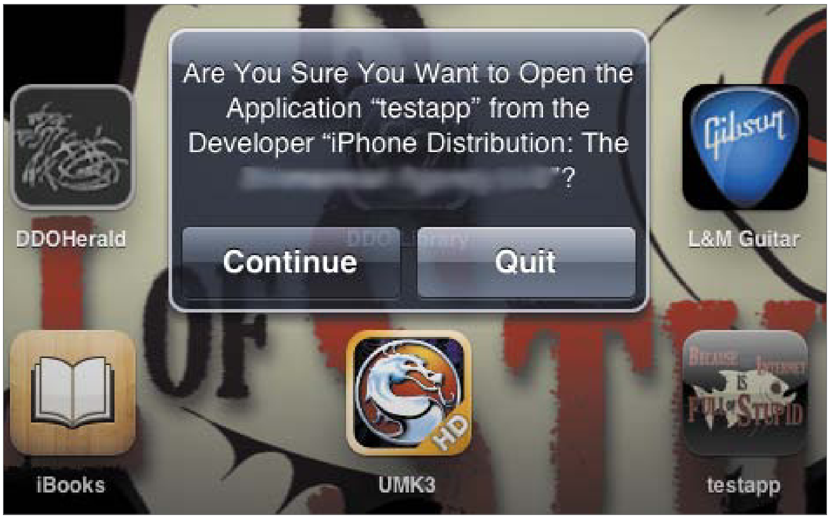


FIGURE 17.6 Application loading

FIGURE 17.7 Application installed



Are You Sure You Want to Open the Application "testapp" from the Developer "iPhone Distribution: The [redacted]"?

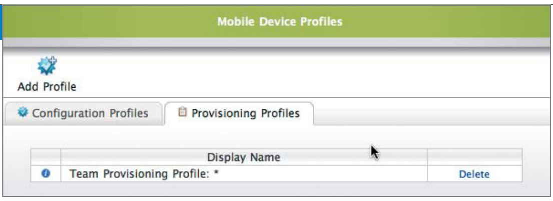
FIGURE 17.8 First-run dialog

18

WIRELESS DISTRIBUTION USING **MOBILE DEVICE MANAGEMENT**

Chapter 17 looked at simple wireless app distribution using just a web server, but that method has some issues. First, the users have to do most of the work: going to the website, downloading the apps, and installing them. Second, it's hard for you to determine if the app was installed. Although Mobile Device Management (MDM) doesn't solve every app distribution problem, it does make managing apps easier, as you'll see in this chapter. (As always, all examples use Casper by JAMF.)

PERFORMING THE **INITIAL SETUP**



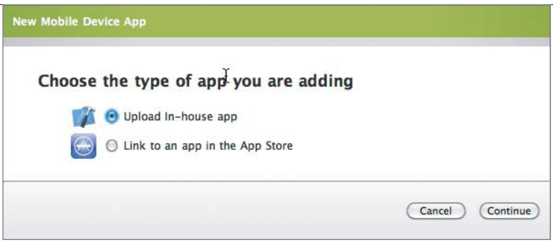

New Mobile Device App
Choose the type of app you are adding
Upload In-house app
Link to an app in the App Store
Cancel
Continue

FIGURE 18.1 Installed Team Provisioning Profile

FIGURE 18.2 Choosing to install an in-house app

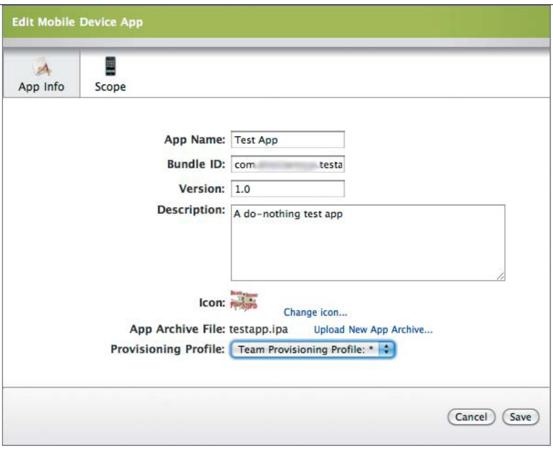
MDM doesn't make all your prerequisites disappear. You'll still need an iOS Developer Enterprise account; you'll still need provisioning profiles, and so on. But, MDM does simplify parts of this process.

Initially, you'll need to set up the provisioning profile for your company under Provisioning in the iOS Provisioning Portal. Then, you'll download the profile and install it in your Mobile Device Management server (**Figure 18.1**).

Then you'll save the app out of Xcode, just as you did in Chapter 17, and add it to your MDM server (**Figure 18.2**).

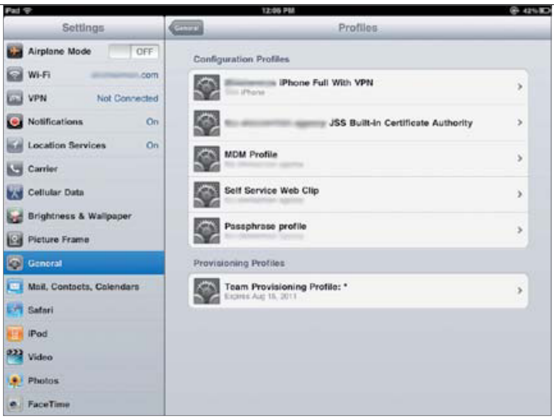
WHAT ABOUT **MAC OS X SERVER 10.7**?

Please note that there is no specific mention of Mac OS X Server 10.7 in this wireless distribution chapter because, to the best of my ability, Apple doesn't actually support wireless in Mac OS X Server 10.7. This makes some sense because as of iOS 4.x, you can't actually push apps to iOS devices. The best you can do is set up a web site from which devices can pull apps (which is how Casper does it). Future versions of iOS may change that, or Apple may support pushing apps in the future. For now, there's nothing in Profile Manager or Mac OS X Server 10.7 for doing so. With Server 10.7, you're on your own.



App Name: Test App
Bundle ID: com.[redacted].testa
Version: 1.0
Description: A do-nothing test app
Icon:
App Archive File: testapp.ipa
Provisioning Profile: Team Provisioning Profile: *

FIGURE 18.3 Information for the in-house app



Airplane Mode: OFF
Wi-Fi: [redacted].com
VPN: Not Connected
Notifications: On
Location Services: On

Configuration Profiles
[redacted] iPhone Full With VPN
[redacted] JSS Built-in Certificate Authority
MDM Profile
Self Service Web Clip
Passphrase profile

Provisioning Profiles
Team Provisioning Profile: *
Expires Aug 18, 2011

FIGURE 18.4 Provisioning profile installed on the device

When you upload the app, you'll need to set up some basic information such as the app name, bundle ID, version, and description. The MDM server also allows you to upload the icon you'll use for the app when distributing it to users.

You'll then upload the app's .ipa file and select the correct provisioning profile to use. Set the scope, or range, of iOS devices that can access this app, and then click Save (**Figure 18.3**).

When you save, the provisioning profile, but not the app, will be uploaded to all devices in the device scope you set (**Figure 18.4**).

INSTALLING THE APP

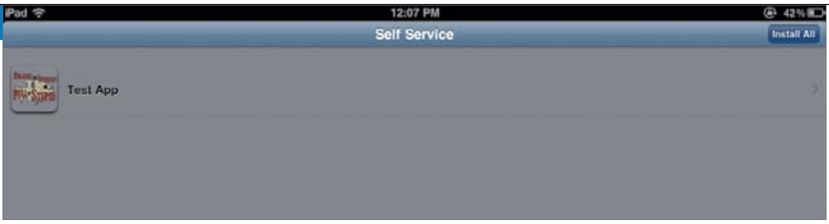


FIGURE 18.5 In-house available app listing

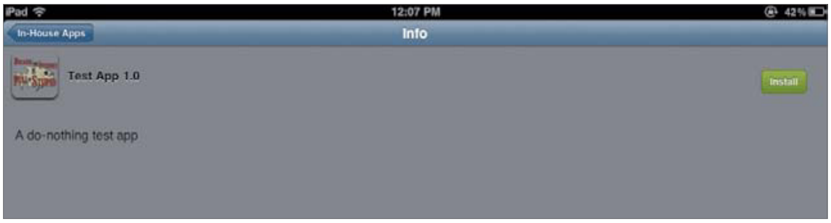


FIGURE 18.6 In-house app ready to install

When the setup is done, the installation can proceed in several ways. In Casper's case, it installs a self-service web application on each iOS device you enroll. Tapping that web application takes you to a webpage that lists available in-house apps, links to App Store apps, and displays updates to in-house apps (**Figure 18.5**).

Tapping the app icon displays the option to install the app and some information about it (**Figure 18.6**).

NOTE: I have no ability to write Objective-C code. This app does nothing but take up space and allow me to install and remove something for the applicable chapters in this book. As you'll see later in this chapter, I have no color sense either.

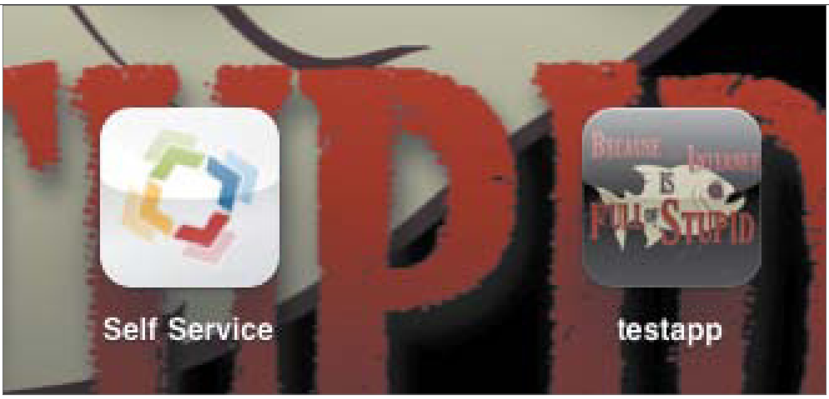
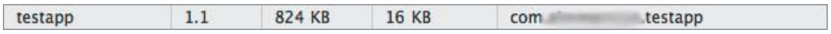


FIGURE 18.7 In-house app installed



testapp	1.1	824 KB	16 KB	com.[redacted].testapp

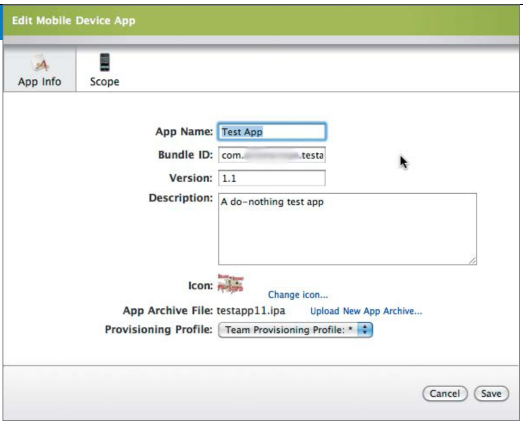
FIGURE 18.8 Inventory for device showing in-house app installed with version number

Tapping Install displays the same "<server name> would like to install <app name>" dialog (**Figure 18.7**) that you see when you wirelessly install any in-house app. Install the app, and you're ready to go.

You may be wondering, "So what? All this has done is save me some work installing provisioning profiles and creating my own webpage." Well, that's partially correct. But the MDM server lets you do one thing that a home-built distribution method can't (at least not as easily): You can check to see if the app has actually been installed (**Figure 18.8**). That's kind of important, and an MDM server is well-suited to letting you know not only if the app has been installed, but which version (which is handy when you have to update the app).

So while this distribution method can't give you the same installation confidence that you get from pushing an app, a decent MDM server does a good job of helping you figure out who's installed what app and which version.

UPDATING AN APP



App Name: Test App
Bundle ID: com.[redacted].testa
Version: 1.1
Description: A do-nothing test app
Icon:
App Archive File: testapp11.ipa
Provisioning Profile: Team Provisioning Profile: *

FIGURE 18.9 Setting up updates to an in-house app

The initial app install is only part of the story. At some point, you're going to update that app, and you'll want to distribute it as easily as you distributed the initial version. MDM also makes this easy. In our example, we've made a minor, yet somewhat terrifying update to our app that we want to inflict upon the users.

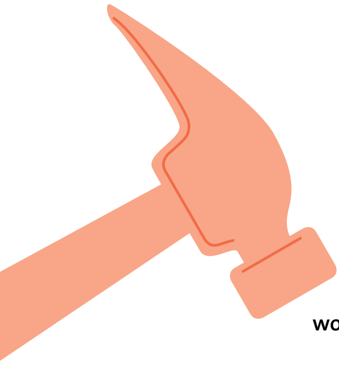
To update the app, go into your MDM server and choose to edit the app you have uploaded. Then upload a new .ipa file, maybe a new icon, and change the version number. (You could also change the icon file and even the app name, if you'd like.)

The important thing *not* to change is the Bundle ID, which allows the MDM server to identify this change as an update to an existing app and not a completely new app (**Figure 18.9**).

By retaining the Bundle ID, the new app version installs over the old one, so you won't have two versions of the same app on every device in your company. That could be a bit confusing.

When you go back into Casper's self-service web application, you'll find an update available (**Figure 18.10**).

Install the update, and you go from an old boring app that (in this case) doesn't do anything (**Figure 18.11**) to an equally unproductive but new, exciting, *colorful* version (**Figure 18.12**).



NOTE: I sincerely hope that the iOS developers you work with have more sane color preferences than I do.

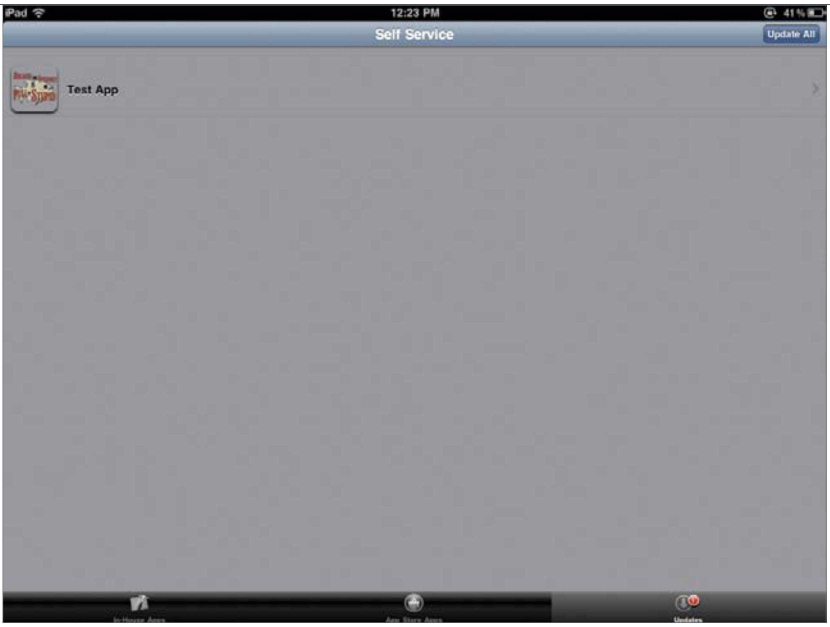
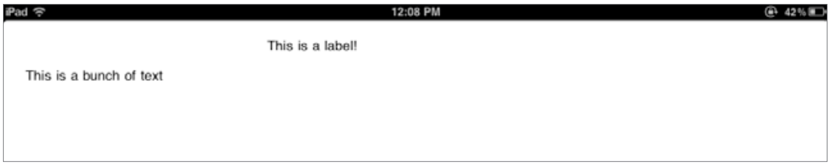


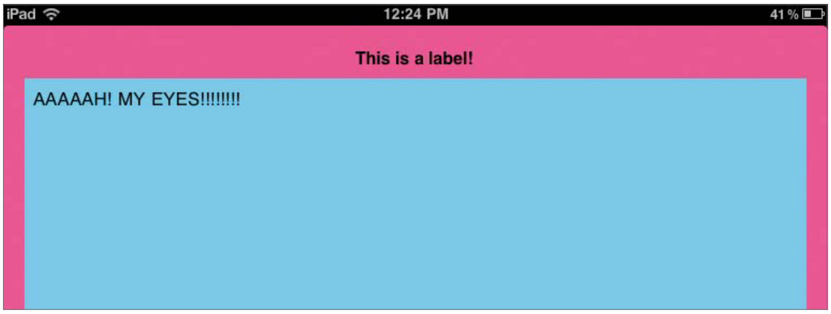
FIGURE 18.10 Available in-house app updates listing



This is a label!

This is a bunch of text

FIGURE 18.11 Version 1.0 of Test App

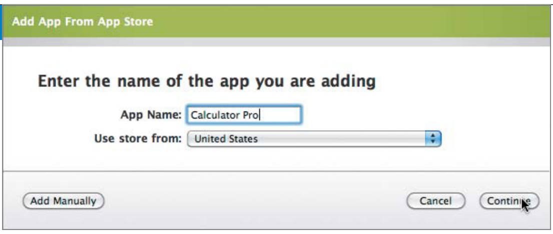


This is a label!

AAAAAH! MY EYES!!!!!!!!

FIGURE 18.12 Version 1.1 of Test App

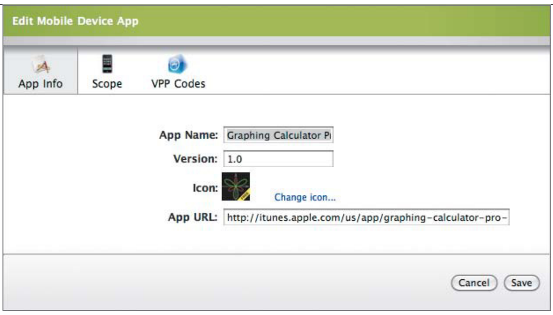
MANAGING **APP STORE** APPS


Add App From App Store

Enter the name of the app you are adding

App Name: Calculator Pro

Use store from: United States


Edit Mobile Device App

App Name: Graphing Calculator P

Version: 1.0

Icon:

App URL: http://itunes.apple.com/us/app/graphing-calculator-pro-

FIGURE 18.13 Setting up a link to an App Store app

FIGURE 18.14 Final information for link to App Store app

Wouldn't it be nice if you could manage App Store apps the same way you manage in-house apps? Say you found an App Store app that's really useful. Wouldn't you like to buy a license for every iOS device in your company, and enable your users to install it quickly and easily? That would be awesome.

Well, if you're in education, you're set. Apple has an App Store Volume Purchase Plan (VPP) just for you. Literally. Just for education. It allows you to buy multiple copies of an app and generate app purchase codes. You distribution the codes to your users, who then go to the App Store, redeem the codes, and install the app. Life is good. If you're not in education, well, you can't do that. Life isn't as good.

You can do *some* things to streamline App Store app management using your MDM server. You can create links to App Store apps that users can then use to simplify the installation process.

The procedure for setting this up is similar to setting up in-house apps (depending on your MDM server).

Set up your Mobile Device Management server to create a link to an App Store app. Enter the name of the app, and the country of the App Store you want to use (**Figure 18.13**).

EDUCATION ONLY? **REALLY?**

I understand that educators, especially in grades K–12, have special needs that don't apply to the business world. For example, I rarely have a lot of five-year-old users. Or 12-year-olds, for that matter. But, you and I and anyone running multiple iOS devices regularly has to face the tedium of installing the same App Store app on multiple iOS devices. Something like the VPP would be *perfect* for us. But it's education-only. Sigh.

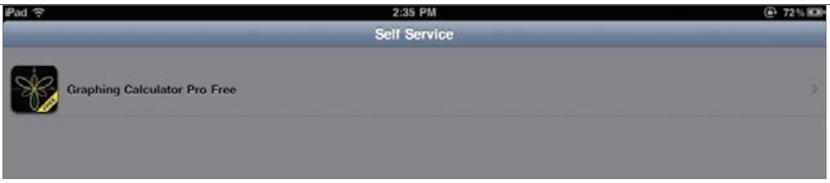
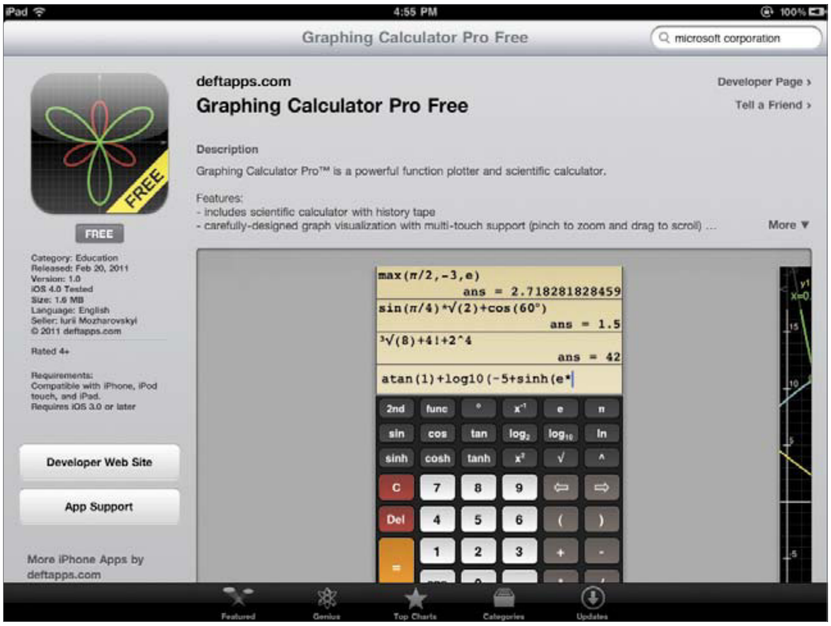


FIGURE 18.15 Available App Store app listing



Description

Graphing Calculator Pro™ is a powerful function plotter and scientific calculator.

Features:
- includes scientific calculator with history tape
- carefully-designed graph visualization with multi-touch support (pinch to zoom and drag to scroll) ...

Category: Education
Released: Feb 20, 2011
Version: 1.0
iOS 4.0 Tested
Size: 1.6 MB
Language: English
Seller: Iurii Mozharovskyi
© 2011 deftapps.com

Rated 4+

Requirements:
Compatible with iPhone, iPod touch, and iPad.
Requires iOS 3.0 or later

FIGURE 18.16 The app's page in the App Store as displayed on the iOS device

The MDM server will search the App Store for the app name and give you a list of results.

Pick the app you want to use, and the MDM server will pull down the actual app name, version, icon, and App Store URL (**Figure 18.14**).

Set up the scope of iOS devices allowed to access this link, and click Save.

When the user opens up the self-service web application, he'll see the app you set up in the App Store section (**Figure 18.15**).

Tapping the app icon will take the user to the app's entry in the App Store, where he can install it (**Figure 18.16**).

WRAPPING **UP**

The method described in this chapter isn't much of an improvement, but at least it lets you present a list of App Store apps for your users, and you can use your MDM server's inventory features to see if they've installed the app.

This page intentionally left blank

19

ISSUES WITH WIRELESS APP **DISTRIBUTION**

As with all technical conveniences, there are some issues to consider with wireless app distribution that may not make it the ideal solution for your environment.

CONSIDERING **INFRASTRUCTURE**

Obviously, to set up wireless app distribution, you have to deal with infrastructure issues. You'll need, at the very least, a web server. Whether you use a cloud service, (in this case, I mean a cloud-based web server such as those provided by Media-Temple or Rackspace), or you maintain the server internally, it's still another web server—even if it's a virtual host on an existing web server. You'll need to address security issues for that server, including access control, versions, patching, and all the other stuff that makes running a server so very much fun.

If you're going to allow server access from outside your network, you'll have another layer of fun. Will you allow only SSL access to the server, or does your situation require some form of VPN connection? Are you going to tie your server into your existing directory service infrastructure, or maintain a separate authentication database specifically for this server?

If you're using Mobile Device Management (MDM), some of these issues are already resolved, but as Chapter 16 pointed out, MDM adds its own layers of complexity to your network.

Oh, and there's also the requirement that certain Apple servers be accessible to install the apps. If your distribution server is outside of your network, that requirement isn't a big deal, but if that server is inside your network, you may be facing a bigger deal than you anticipated.

ADDING **ISSUES** FOR **DEVELOPERS**

iOS developers will have even more steps added to the setup process because readying the application for distribution now involves incorporating web server data and creating extra copies of icon files. There's also a *development cost* in terms of the app distribution interface offered to users. Do you just show them a page of assorted links, or like Casper, do you separate out new apps and app updates to individual pages? The latter is obviously nicer for users, and highly recommended; but, it increases the complexity of interacting with the server on the back end.

If you have separate iPhone/iPod Touch and iPad apps, do you leave navigation between those versions up to the user? Do you try to sniff out which device is in use so that the user sees only those apps that she can run? Again, the former is easier on you but adds to the number of "Oops, I installed the wrong app" calls. The latter is easier on the users but harder on IT.

ADDRESSING **APP MANAGEMENT**

Then there's the Apple App Store issue. Do you try to provide links to "approved" apps on your distribution server, or do you just rely on the users to install those? What if they need to pay for those apps? Do they do so using a company account that's the same on every device, or do they use their own accounts? This situation becomes even *more* fun if you are trying to restrict the App Store apps that they can install on their devices. Restricting app installs requires a binary setting at present, so if you disallow free installation of apps, wireless app distribution gets really . . . well, *interesting* from a management point-of-view.

What if you have to delete an app? MDM servers such as Casper make parts of this process easy, like removing the provisioning profile, and removing the app from the self-service web clip. But reaching out and removing apps from iOS devices is not something that anyone (but *maybe* Apple) can do.

DELETING YOUR OWN **STUFF**

I hope Apple pays attention to this area in the next major version of iOS. Yes, I can see why they might not want to allow you to just delete <random app> from a device; but, for enterprise apps, it is silly to not allow users to quickly and easily remove enterprise apps from devices belonging to the company. Of course, if the company is paying for an App Store app, even on a personal device, when that person leaves the company, it should be possible for the company to remove (only) the apps that the company paid for. This is why a solution like the VPP for enterprise would be a good thing. It could enable better management of devices so that company-purchased apps or company-written apps could be proactively and positively removed from devices as necessary.

WRAPPING **UP**

I really, really don't want to scare anyone away from wireless app distribution. Having had to distribute apps the other way, serving remote users in less than stellar conditions, I don't even think this is a scaling issue. For in-house apps, wireless distribution is absolutely better, as far as I'm concerned.

But, wireless app distribution does bring issues that must be managed and dealt with, whether you want to or not. It's better to think about them and plan for them *before* you try to roll out a distribution server (or worse, a few months *after* you've rolled out that server). Wireless app distribution is awesome, and it'd be a shame not to use it because of some picayune issue that could have been handled before the process blew up in your face.

INDEX

WATCH
READ
CREATE