Intune MAM Quick Start Guide

Managing LOB apps without device enrollment

Prepared by Microsoft PM Olivia Dang (olivia.dang@microsoft.com)

Introduction

With the GA of the Intune App Wrapping Tool for MAM without device enrollment, we've written this "quick start" guide to help you experience an example of deploying an Intune-managed line-of-business (LOB) app to a device without requiring device enrollment. We'll guide you through the following high-level workflow:

1. **Enable MAM capabilities on the LOB app "Wagr," using the Intune App Wrapping Tool**.
2. **Author and target MAM policy Wagr without requiring device enrollment.** We'll use the Azure console to define and target MAM policies.
3. **Deploy the app to end-user devices.** For now, we must sideload the app to a test device. For MAM without device enrollment, you should *not* use the classic Intune management portal (manage.microsoft.com) to install the managed app to a device.

Wrap the LOB app

We'll go through the process of using the App Wrapping Tool on an open-source iOS app called Wagr and a simple Android app called HelloWorld. Wagr is an example of an internal LOB app that an IT department may want to manage on employees' devices.

Wrapping iOS App Wagr

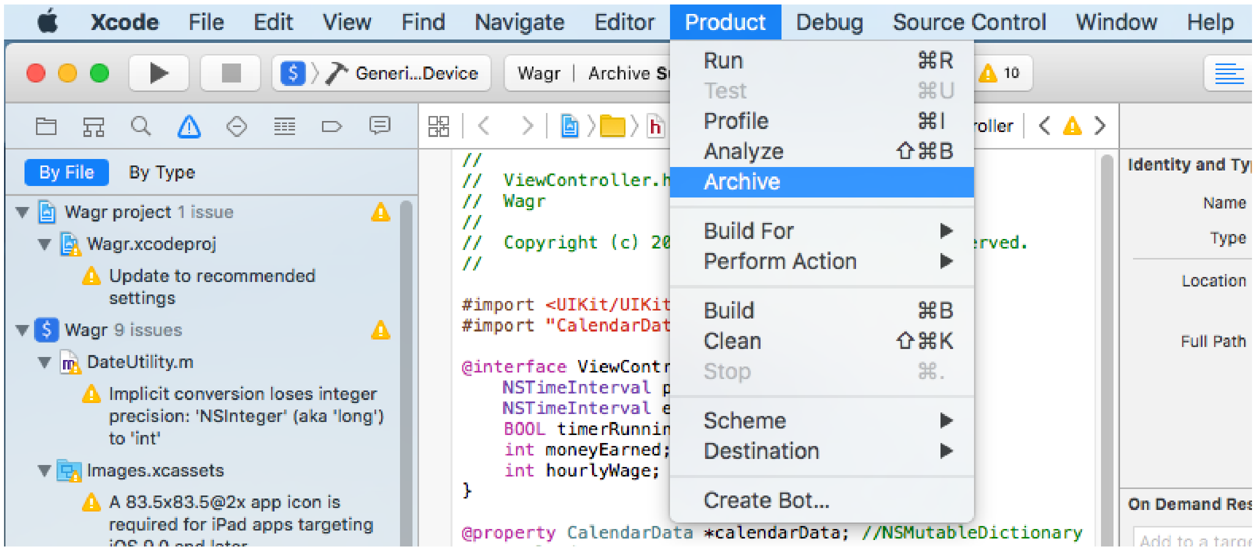
For the full and official documentation, please visit the Microsoft documentation page for the Intune App Wrapping Tool for iOS.

Prerequisites

1. A Mac computer that runs OS X 10.8.5 or later, which has the Xcode toolset version 5 or later installed.
2. A valid Apple signing certificate and provisioning profile. See your Apple developer documentation and this Microsoft blog post for help on how to obtain a signing certificate and provisioning profile.

Build Wagr.ipa

Download Wagr from GitHub. Open the Xcode project. In the Xcode options, choose Product → Archive.



Update to recommended settings

Implicit conversion loses integer precision: 'NSInteger' (aka 'long') to 'int'

A 83.5x83.5@2x app icon is required for iPad apps targeting iOS 9.0 and later

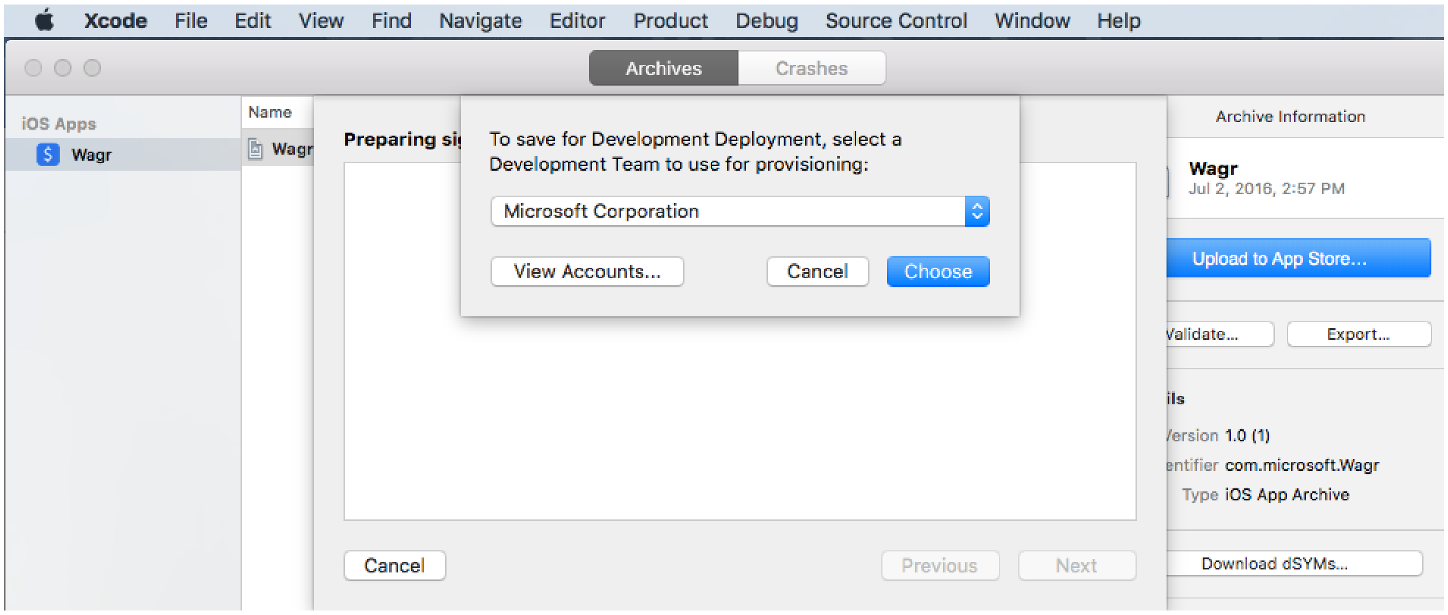
```
//
//  ViewController.h
//  Wagr
//
//  Copyright (c) 20                    rved.
//

#import <UIKit/UIKit
#import "CalendarDat

@interface ViewContr
    NSTimeInterval p
    NSTimeInterval e
    BOOL timerRunnin
    int moneyEarned;
    int hourlyWage;
}

@property CalendarData *calendarData; //NSMutableDictionary
```

In the Archives dialogue, press the "Export…" button in the right-hand pane. Then select "Save for Development Deployment," the last option, as the method for export. Finally, select your development team's signing asset for provisioning. Click "choose."



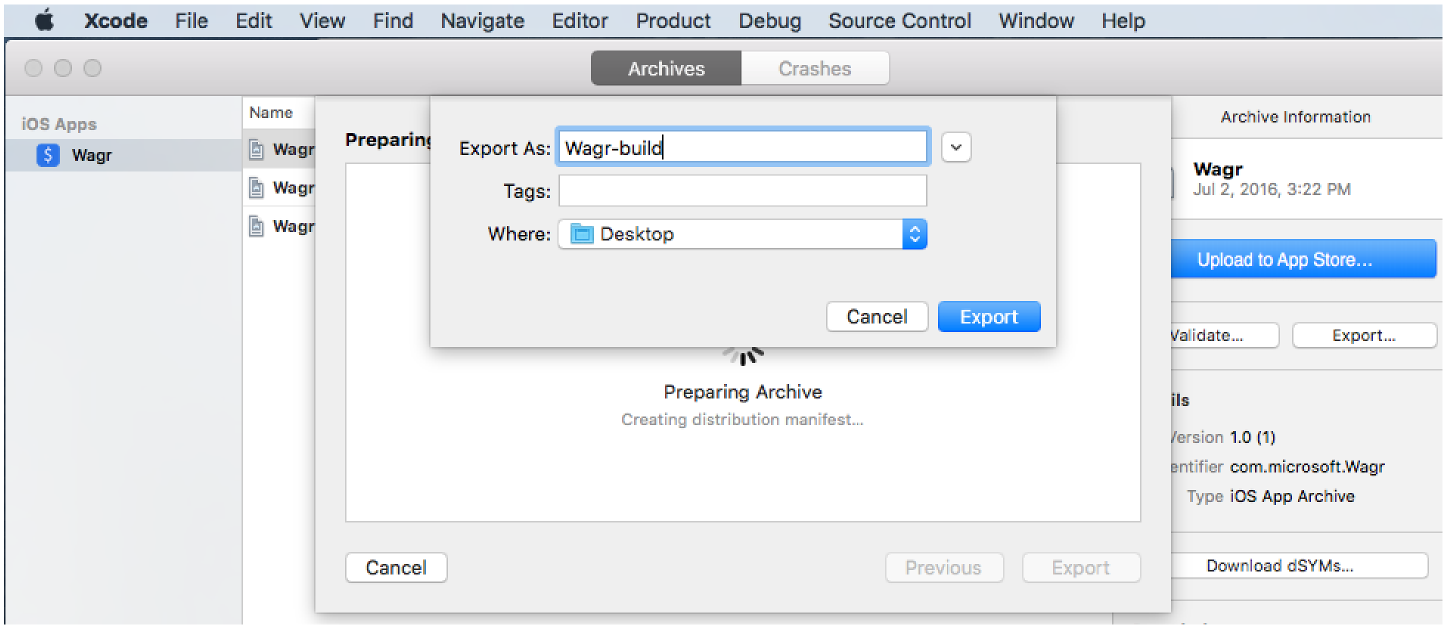
To save for Development Deployment, select a Development Team to use for provisioning:

Microsoft Corporation

Wagr
Jul 2, 2016, 2:57 PM

/ersion 1.0 (1)
entifier com.microsoft.Wagr
Type iOS App Archive

Select the option "Export one app for all compatible devices" and press next. Look over the export summary, then press next. We've selected the directory **~/Desktop/Wagr-build** to contain our exported .ipa file.



Voila! Wagr.ipa should be in the folder **~/Desktop/Wagr-build**. This is the file that contains the built Wagr app.

Prepare Mobile Provisioning Profile and Obtain SHA1 Hash of Signing Certificate

To obtain the signing certificate and mobile provisioning profile you need from your company, read this helpful Intune blog post.

In this example, my signing certificate was downloaded from the Apple Developer portal. I made sure to add it to my Mac OS X Keychain by opening the certificate file.

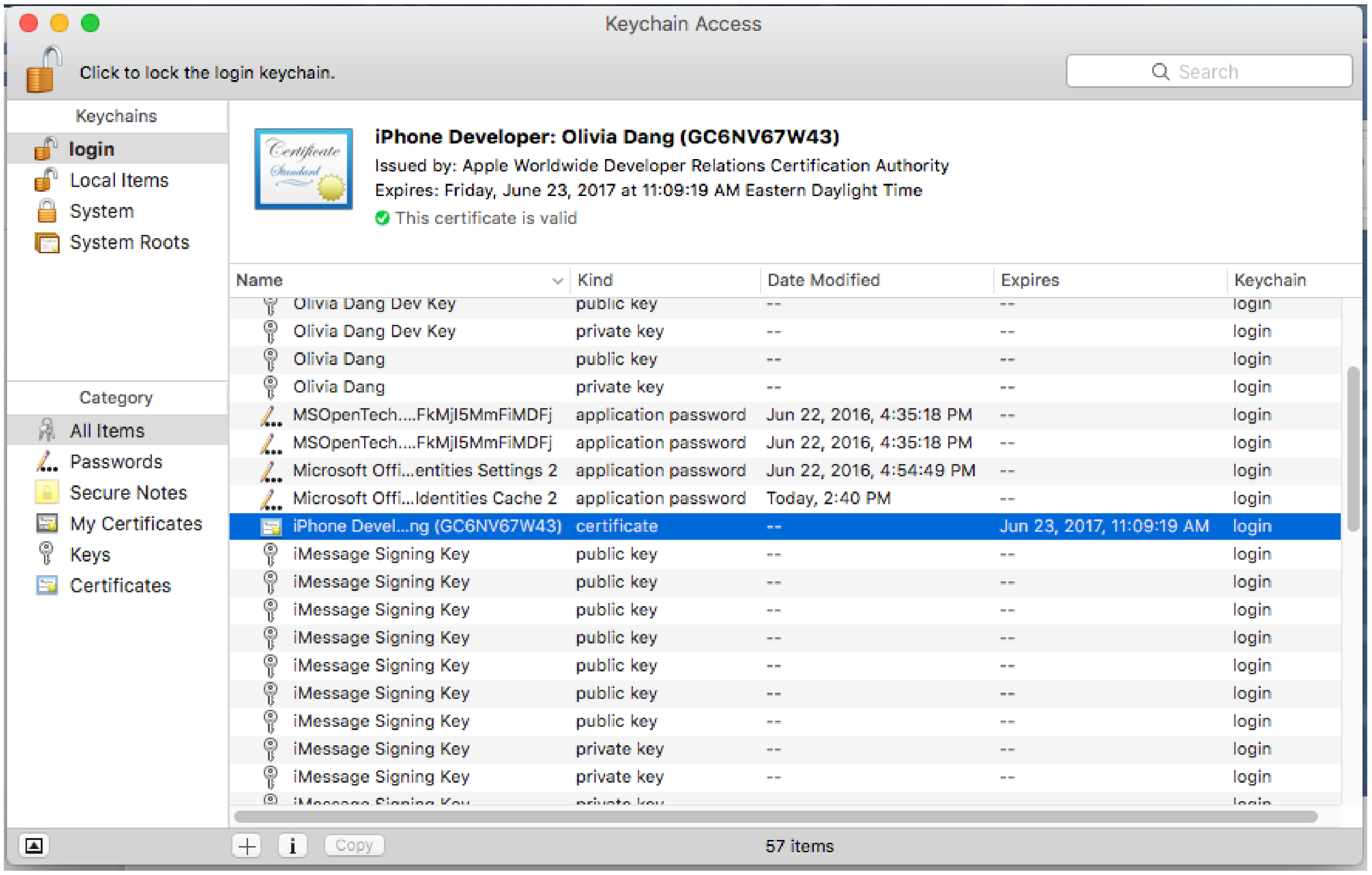
In this example, my provisioning profile was downloaded from the Apple Developer portal and contains the device ID of the iOS device onto which we will install Wagr.

Note: **Because this is in an internal test environment, the signing certificate is a developer certificate, and the test device ID must be in the provisioning profile.**

I've placed the provisioning profile on my Desktop.

Finder File Edit View Go Window Help
PROV
iOS_Team_Provisioning_Pro...rovision

To get the SHA1 hash of the signing certificate, open Keychain Access in OS X Settings to the certificate.



iPhone Developer: Olivia Dang (GC6NV67W43)
Issued by: Apple Worldwide Developer Relations Certification Authority
Expires: Friday, June 23, 2017 at 11:09:19 AM Eastern Daylight Time
This certificate is valid

Name	Kind	Date Modified	Expires	Keychain
Olivia Dang Dev Key	public key	--	--	login
Olivia Dang Dev Key	private key	--	--	login
Olivia Dang	public key	--	--	login
Olivia Dang	private key	--	--	login
MSOpenTech....FkMjI5MmFiMDFj	application password	Jun 22, 2016, 4:35:18 PM	--	login
MSOpenTech....FkMjI5MmFiMDFj	application password	Jun 22, 2016, 4:35:18 PM	--	login
Microsoft Offi...entities Settings 2	application password	Jun 22, 2016, 4:54:49 PM	--	login
Microsoft Offi...Identities Cache 2	application password	Today, 2:40 PM	--	login
iPhone Devel...ng (GC6NV67W43)	certificate	--	Jun 23, 2017, 11:09:19 AM	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	public key	--	--	login
iMessage Signing Key	private key	--	--	login
iMessage Signing Key	private key	--	--	login
iMessage Signing Key	private key			login

Double-click on the certificate to open a dialog with its information. Scroll to the bottom of the dialog and you'll find, under "Fingerprints," the SHA1 hash. Copy and paste this string somewhere handy because it is a required parameter to the wrapping command. An example SHA1 hash string looks like:

`"12 A3 BC 45 D6 7E F8 90 1A 2B 3C DE F4 AB C5 D6 E7 89 0F G2"`

Install the App Wrapping Tool for iOS

Download and the iOS App Wrapping Tool from GitHub onto a Mac OS machine. Read and accept the End User License Agreement.

Run the .dmg installation file, which will mount the contents of the App Wrapping Tool, and copy the contents to a local directory.

Run the App Wrapping Tool for iOS

Open a Terminal window navigate to the folder where you saved the contents locally. Navigate to the subdirectory **`/Contents/MacOS`.**

Option 1: Run with command line arguments

To wrap Wagr.ipa with command line arguments, we run the following command:

```
./IntuneMAMPackager -i ~/Desktop/Wagr-build/Wagr.ipa -o ~/Desktop/Wagr_wrapped.ipa -p
~/Desktop/iOS_Team_Provisioning_Profile_.mobileprovision -c "12 A3 BC 45 D6 7E F8 90 1A
2B 3C DE F4 AB C5 D6 E7 89 0F G2" -v
```

Below is a complete summary of the command line arguments with the iOS app wrapping tool:

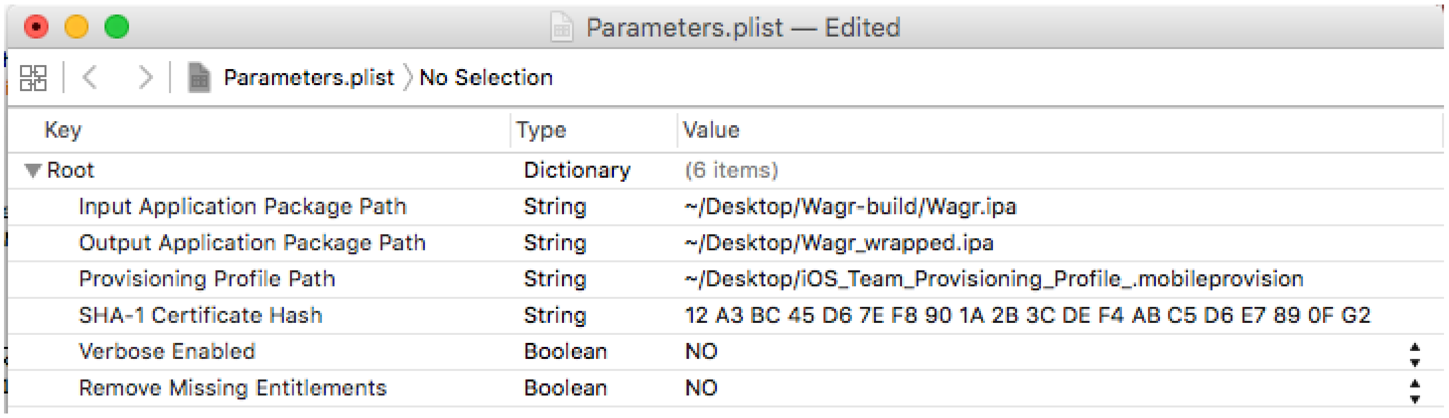
Property	Information
-h	Displays the available command line properties for the app wrapping tool and example usages. [strongly encourage to take a look at this by running the command `./IntuneMAMPackager -h`]
-i	<Path to the input native iOS application or previously packaged iOS application>
-o	<Path to the output application>
-p	<Path to the provisioning profile>
-c	<SHA1 hash of the certificate>
-e	(Optional) Remove missing entitlements during packaging
-xe	(Optional) Prints information about the iOS extensions in the app, and what entitlements are required to use them. See the "Setting app entitlements" section for more details.
-x	(Optional) <An array of paths to extension provisioning profiles>. Use this if your app needs extension provisioning profiles.
-v	(Optional) Output verbose messages to the console.
-f	(Optional) <Path to a plist file specifying arguments>
-b	(Optional) Leave blank if you want the wrapped app to have the same Bundle Version as the input app (not recommended). Specify <custom bundle version> if you want the wrapped app to have a custom Bundle Version. We recommend incrementing the native app's Bundle Version by the least significant component, eg. 1.0.0 -> 1.0.1

Option 2: Run with plist arguments

Another option is to put all our command arguments into a plist file. Plist is a file format similar to XML that can help us input our command-line arguments into a key-value interface.

In **`/Contents/MacOS`**, open `Parameters.plist`, a blank plist template, with a text editor or Xcode.

Enter our arguments for input path, output path, provisioning profile path, SHA1 certificate hash, and verbose enabled. Our resulting plist looks like this:



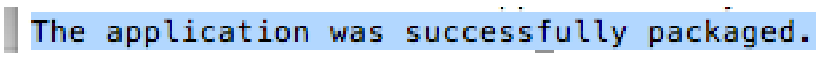
Key	Type	Value
▼ Root	Dictionary	(6 items)
Input Application Package Path	String	~/Desktop/Wagr-build/Wagr.ipa
Output Application Package Path	String	~/Desktop/Wagr_wrapped.ipa
Provisioning Profile Path	String	~/Desktop/iOS_Team_Provisioning_Profile_.mobileprovision
SHA-1 Certificate Hash	String	12 A3 BC 45 D6 7E F8 90 1A 2B 3C DE F4 AB C5 D6 E7 89 0F G2
Verbose Enabled	Boolean	NO
Remove Missing Entitlements	Boolean	NO

Finally, run the IntuneMAMPackager with the plist as an argument:

```
./IntuneMAMPackager –f Parameters.plist
```

Success!

Both options of running the IntuneMAMPackager do the same thing. They output a wrapped app in our Desktop named Wagr_wrapped.ipa. If everything succeeded, terminal should output:



```
The application was successfully packaged.
```

We've officially wrapped our LOB app Wagr! The wrapped version is Wagr_wrapped.ipa, and is located in the output folder we specified.

Wrapping the Android App HelloWorld

For full documentation, please visit the Intune documentation page for the App Wrapping Tool for Android.

Prerequisites

1. A Windows machine running Windows 7 or later.
2. An LOB app developed by, or for your company. You cannot use the app wrapper on apps downloaded from the Google Play Store. The app must be a valid Android app package with the extension **.apk** file and:
    a. Cannot be encrypted
    b. Must be written for Android 4.0 or later
3. The latest version of the Java Runtime Environment. Ensure that the Java path variable has been set to: **`C:\ProgramData\Oracle\Java\javapath`**

Use the Java Key Tool to Generate Key Pair

Android requires all .apks to be signed before deployment. Before wrapping the app, you'll need to generate a key pair (a public key and associated private key) using the Java Key Tool.

Navigate to the directory of the Key Tool. It looks something like this, depending on the latest version of the JRE:

`C:\Program Files (x86)\Java\jre1.8.0_91\bin`

For example, the following command generates a key pair with:

1. The keystore located in a file called *keystorefile*
2. Generated with RSA encryption algorithm
3. A key size of 1024
4. A validity period of 3,000 days

```
keytool.exe -genkeypair -v -keystore keystorefile -keyalg RSA -keysize 1024 -validity
3000
```

Install the App Wrapping Tool

Download the Intune App Wrapping Tool for Android from GitHub. Run **InstallAWT.exe** to install the app wrapping tool. By default, the tool is installed to the directory:

`C:\Program Files (x86)\Microsoft Intune Mobile Application Management\Android\App Wrapping Tool`

Run the App Wrapping Tool for Android

In the installation directory, open a PowerShell window in Administrator mode. Import the app wrapping tool PowerShell module using the following command:

```
Import-Module .\IntuneAppWrappingTool.psm1
```

To view help for the tool (which includes syntax, example usages, and technical information), run the command:

```
Help Invoke-AppWrappingTool
```

Run the app wrapping tool on the HelloWorld.apk file by using the **invoke-AppWrappingTool** command together with the following parameters.

```
invoke-AppWrappingTool -InputPath HelloWorld.apk -OutputPath HelloWorld_wrapped.apk -
KeyStorePath "C:\Program Files (x86)\Java\jre1.8.0_91\bin\keystorefile" -keyAlias ks -
SigAlg SHA1withRSA -Verbose
```

PowerShell will then prompt us for the `-KeyStorePassword` and `-KeyPassword` arguments, since we didn't specify it in the command above. Enter the password you generated with the Java Key Tool.

Voila! You should have the wrapped application called HelloWorld_wrapped.apk in the working directory.

Below is a full description of parameters for the command:

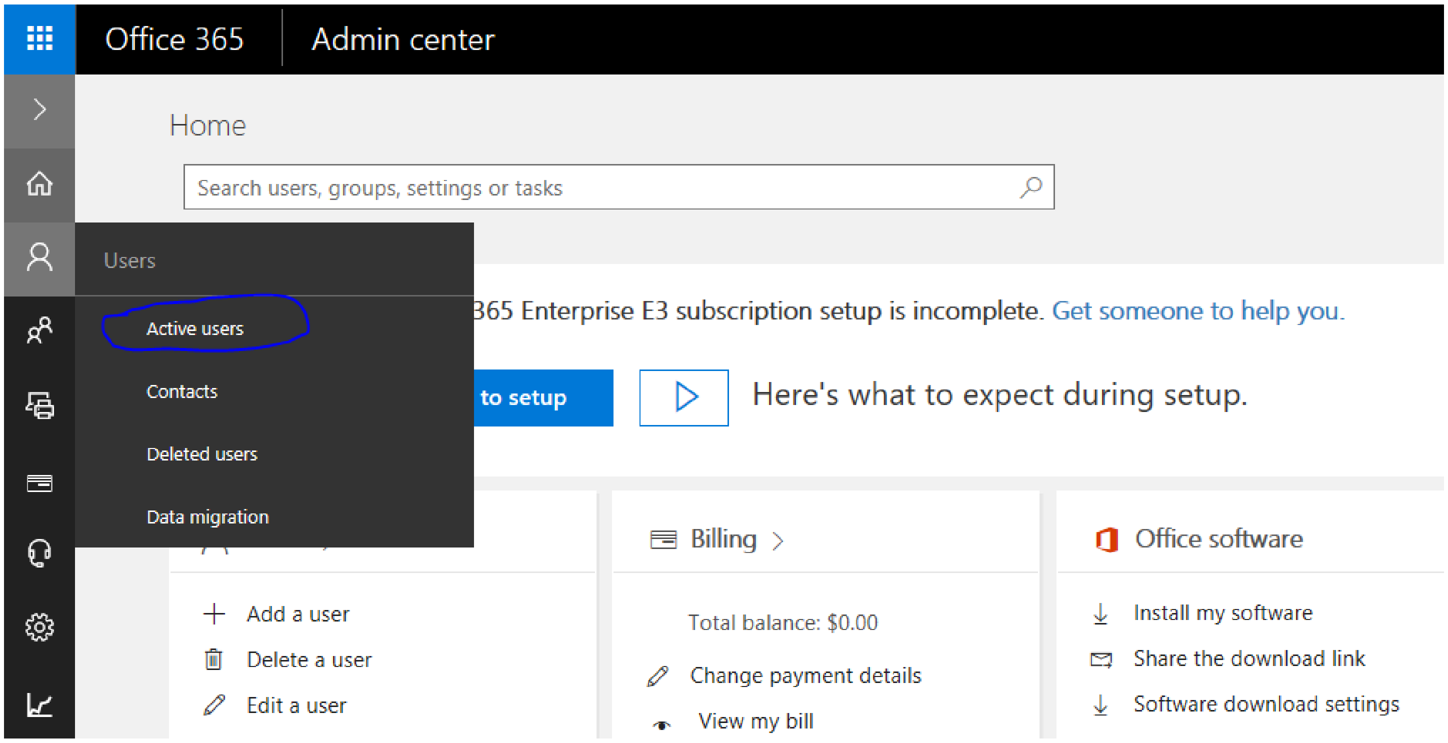
Property	Information
-InputPath	<Path to the input Android application>
-OutputPath	<Path to the output Android application. If this is the same directory path as InputPath, the packaging will fail.>
-KeyStorePath	<Path to the keystore file containing the public/private key pair for signing the Android app.>
-KeyStorePassword	<Password to decrypt the keystore, generated when you ran the Java Key Tool.>
-KeyAlias	<Name of the key to be used for signing>
-SigAlg	(Optional) <Name of the signature algorithm to be used for signing.> The algorithm must be compatible with the private key. Examples: SHA256withRSA, SHA1withRSA, MD5withRSA
-Verbose	(Optional) Include this if you want detailed output to the PowerShell console
-Debug	(Optional) Include this if you want detailed debugging output to PowerShell

Author and Target MAM Policy to the App without Enrollment

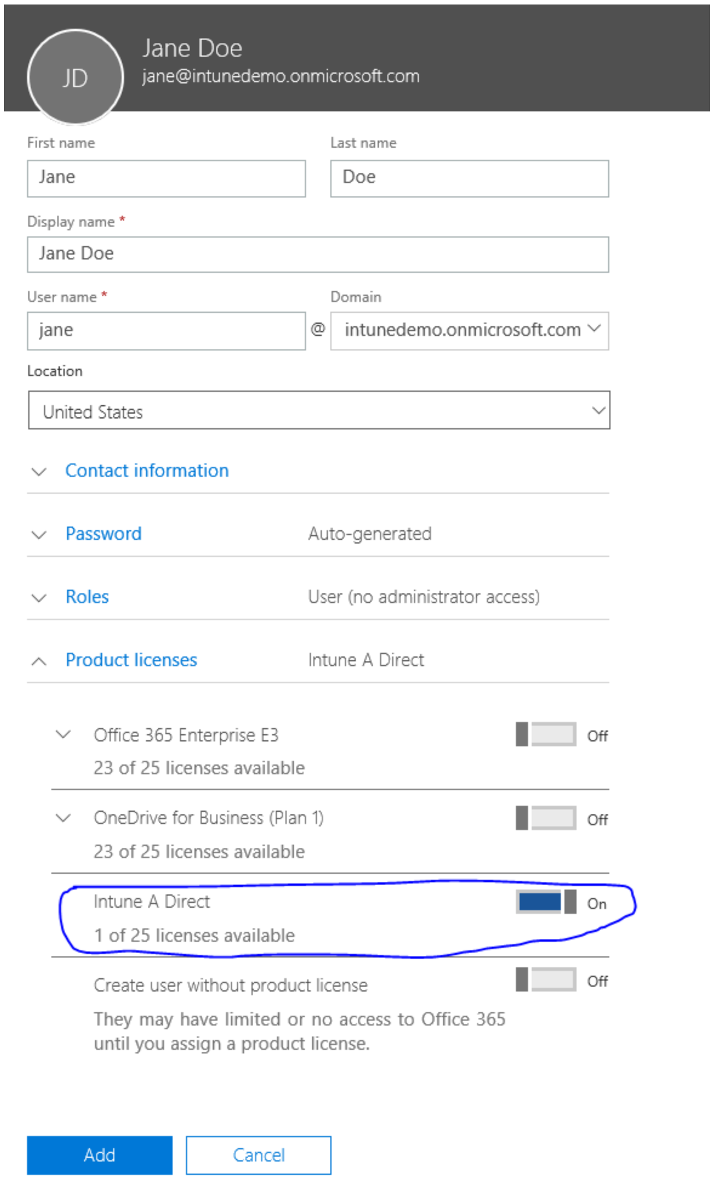
To do anything with Intune, we'll need an Intune administrator account. You can obtain one by visiting the Microsoft Intune website and signing up for a 30-day trial of Intune.

After signing up for the free trial, navigate to the Office 365 Portal, which is the hub for creating users and assigning them to security groups.

Let's add a user to our organization named Jane Doe. On the left-side menu, hover over "Users" and click "Active Users."



Click the **"+ Add a user**" button to create a new user. Fill out the information for your user account. Here we're creating an account for Jane Doe. **Make sure to assign Jane an Intune A Direct license!**


JD

Jane Doe

jane@intunedemo.onmicrosoft.com

First name: Jane

Last name: Doe

Display name *: Jane Doe

User name *: jane @ Domain: intunedemo.onmicrosoft.com

Location: United States

Contact information

Password: Auto-generated

Roles: User (no administrator access)

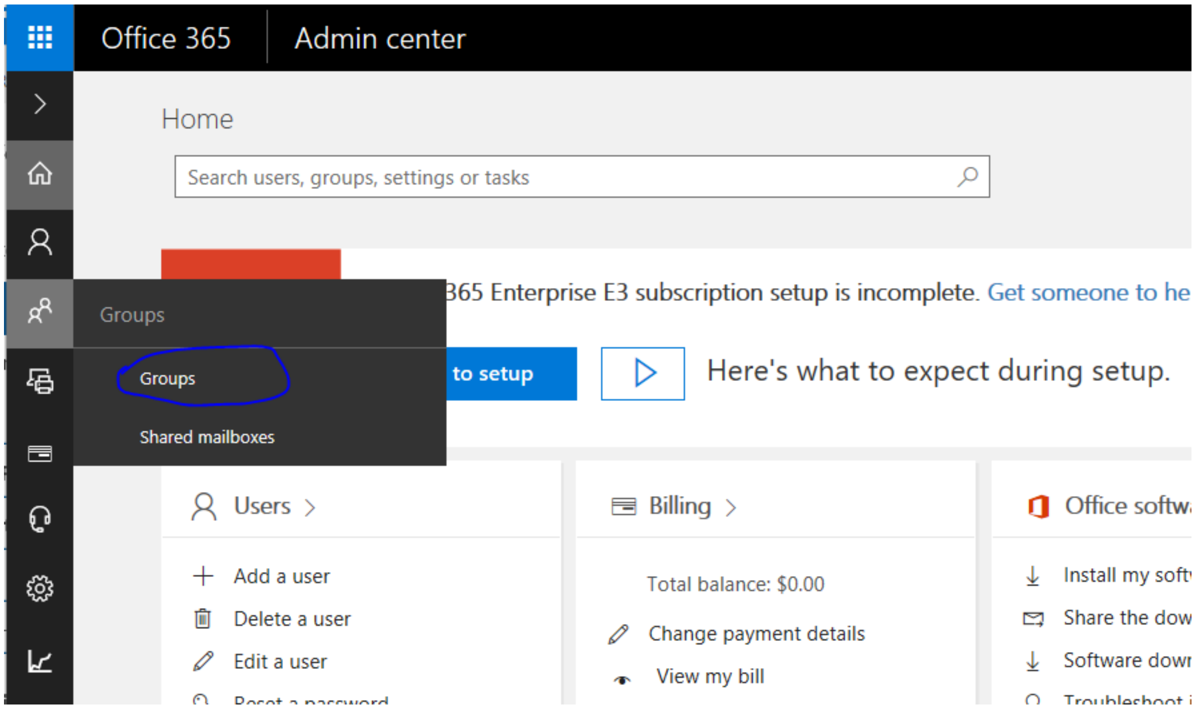
Product licenses: Intune A Direct

Office 365 Enterprise E3 23 of 25 licenses available	Off
OneDrive for Business (Plan 1) 23 of 25 licenses available	Off
Intune A Direct 1 of 25 licenses available	On
Create user without product license They may have limited or no access to Office 365 until you assign a product license.	Off

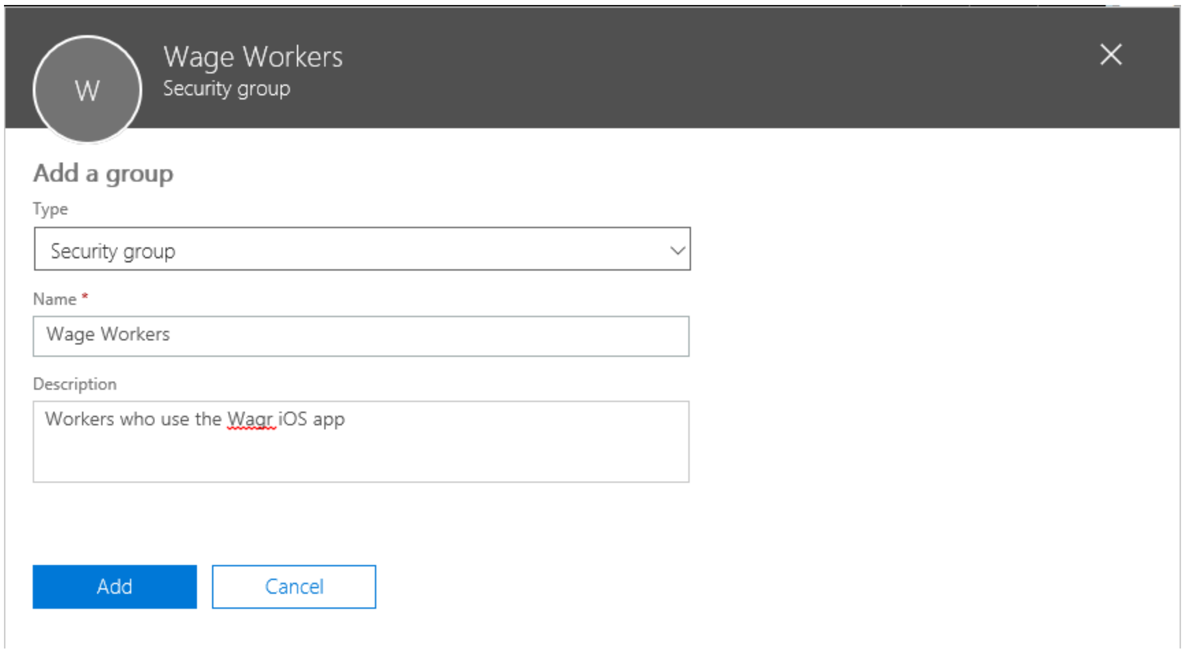
Press “Add” to finalize Jane’s creation.

As the administrator, you’ll receive Jane’s password information. Keep Jane Doe’s password handy – these are her corporate credentials you can use to test your managed apps on an end-user device.

Next, we should create a security group and place Jane into it. The target of a MAM policy can be one or multiple security groups. Click “Groups” on the main menu.



On the left-hand menu, click “Groups.” Then click the **“+ Add a group**” button to create a new security group.



Wage Workers
Security group

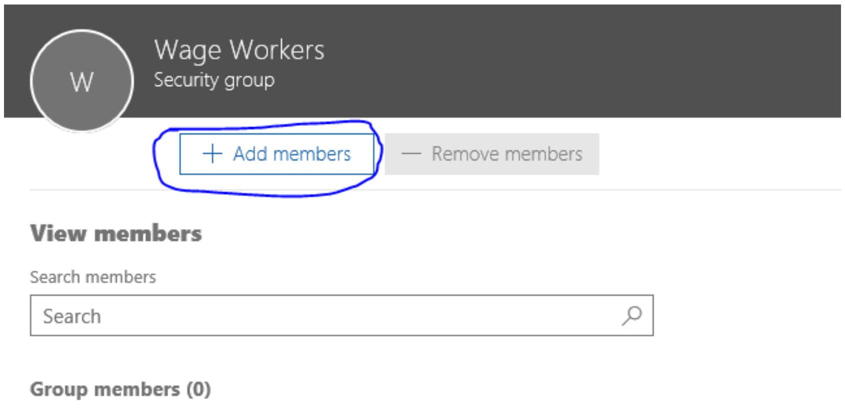
Add a group

Type: Security group

Name *: Wage Workers

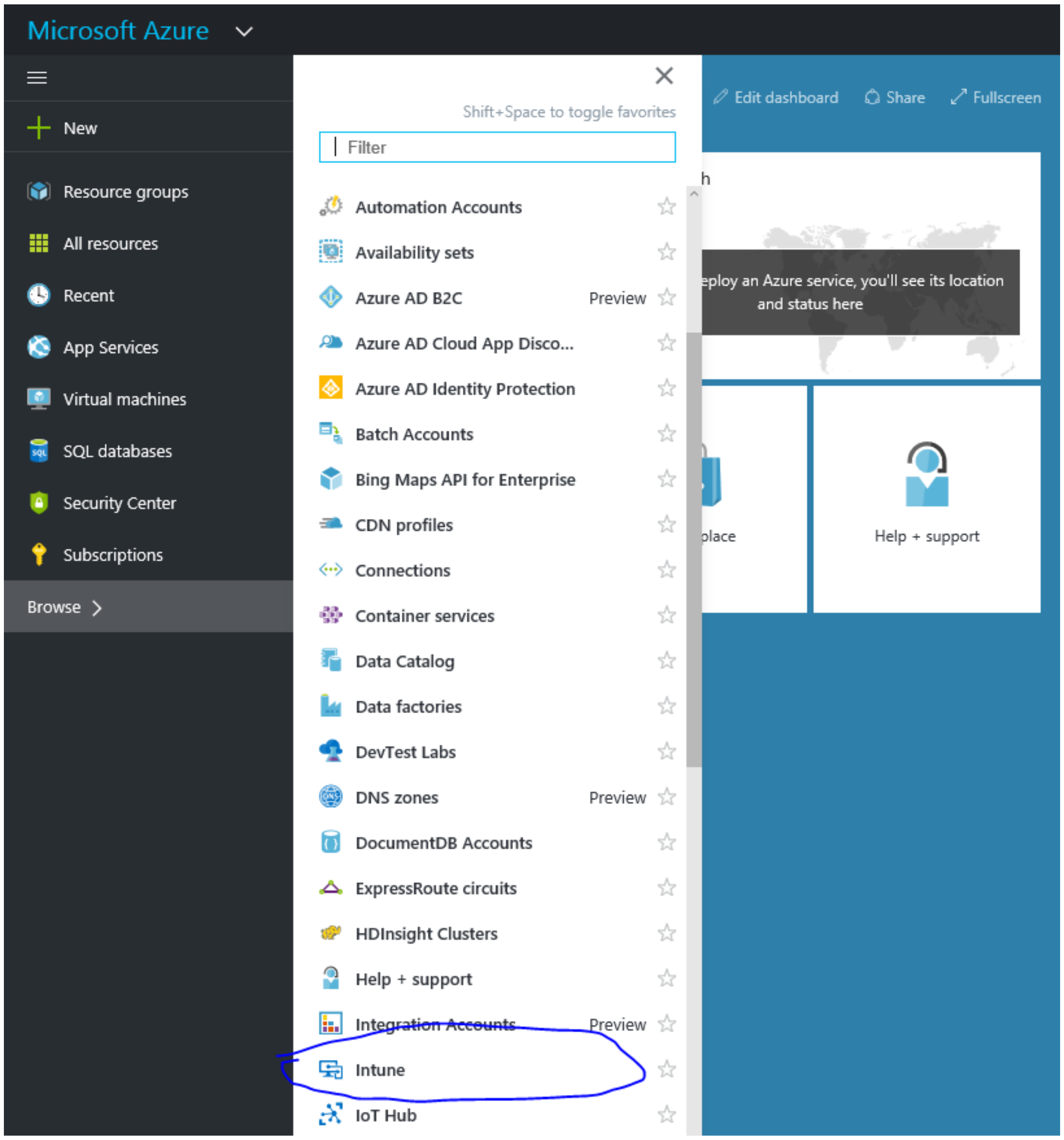
Description: Workers who use the Wagr iOS app

After creating the group, click on the group in the overall list of groups. Edit the membership list and add Jane Doe to the group.

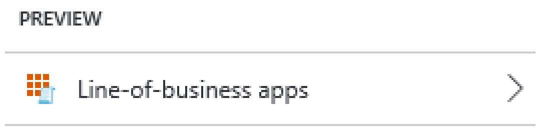


After creating a security group with our user Jane in it, now we switch to using the [Azure Console](), a separate administrator's portal from the Office Portal, to define and target Intune MAM policy.

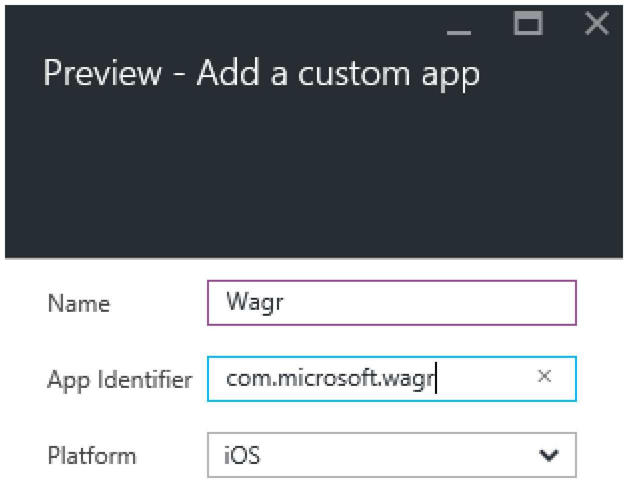
On the left-side menu, click "Browse" to open the list of services manageable from the Azure console. Scroll down to "Intune" and click.



Next, in the right-most "Settings" panel, you'll see a menu of Intune capabilities. Click on:



Here we'll add our wrapped LOB app(s) to the Intune MAM service by their app identifiers. For iOS, the app identifier is the bundle identifier. For Android, it's the package name. For our example, Wagr's bundle id is *com.microsoft.wagr*.



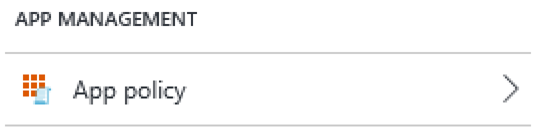
Preview - Add a custom app

Name: Wagr

App Identifier: com.microsoft.wagr

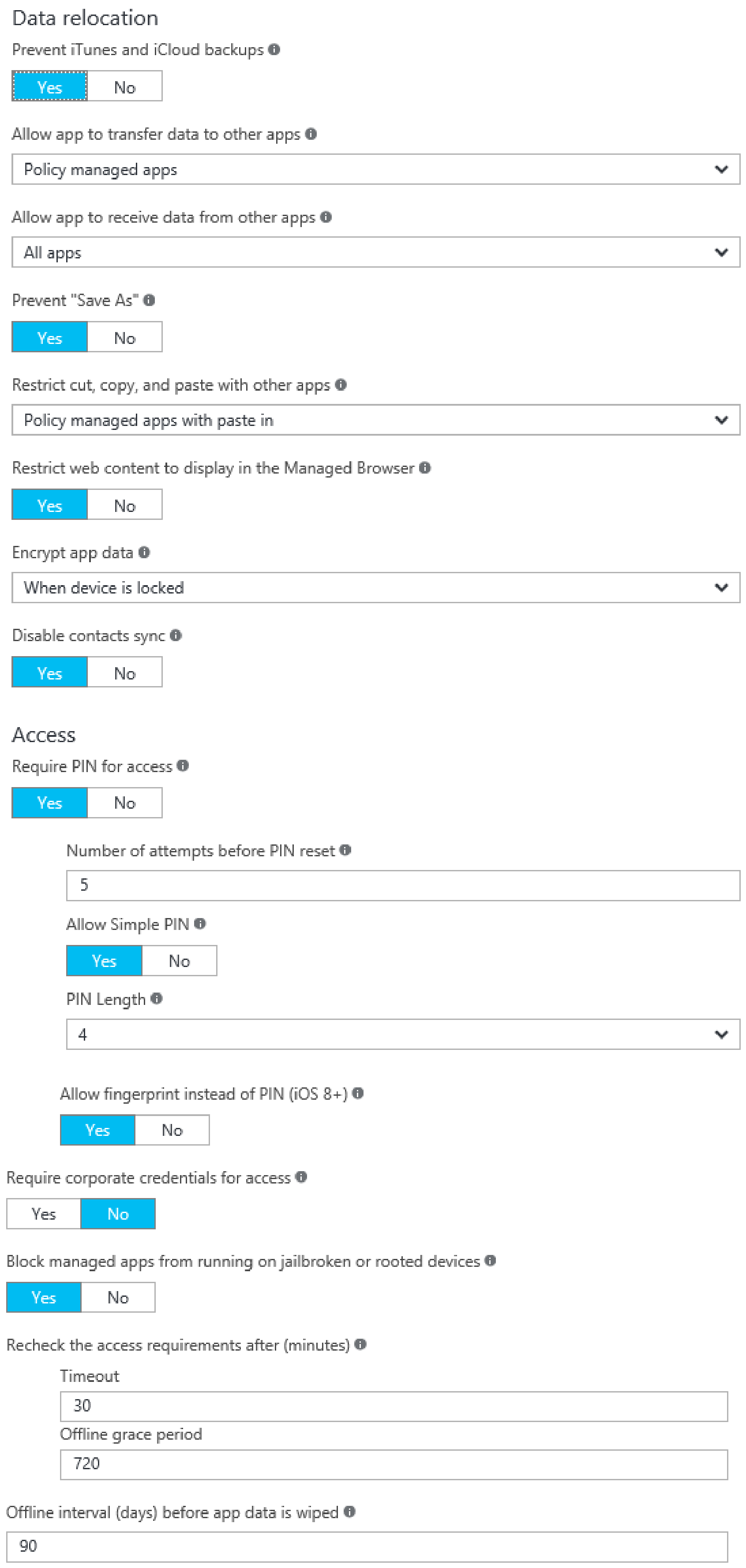
Platform: iOS

Return to the Settings panel in the Intune dashboard. Click on:



Add a new MAM policy for Wagr. Give it a name, description, choose iOS as the platform, and select Wagr as a targeted app. We'll name ours "Olivia Wagr policy." Then click "Configure required settings."

These are all the items you can manage on an app with MAM-without-enrollment.



Data relocation

Prevent iTunes and iCloud backups

[✓ Yes] No

Allow app to transfer data to other apps

Policy managed apps

Allow app to receive data from other apps

All apps

Prevent "Save As"

[✓ Yes] No

Restrict cut, copy, and paste with other apps

Policy managed apps with paste in

Restrict web content to display in the Managed Browser

[✓ Yes] No

Encrypt app data

When device is locked

Disable contacts sync

[✓ Yes] No

Access

Require PIN for access

[✓ Yes] No

Number of attempts before PIN reset

5

Allow Simple PIN

[✓ Yes] No

PIN Length

4

Allow fingerprint instead of PIN (iOS 8+)

[✓ Yes] No

Require corporate credentials for access

Yes [✓ No]

Block managed apps from running on jailbroken or rooted devices

[✓ Yes] No

Recheck the access requirements after (minutes)

Timeout

30

Offline grace period

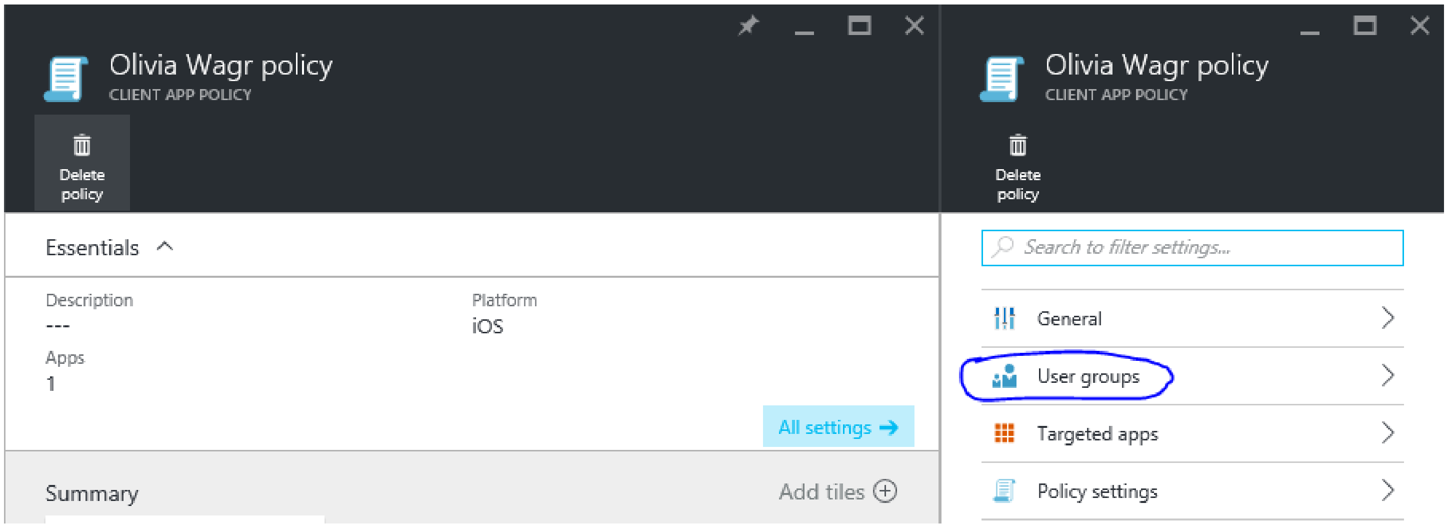
720

Offline interval (days) before app data is wiped

90

Choose the settings you'll enforce in this MAM policy. Finish creating the policy when you're done.

To deploy the policy, you must target at least one security group of users. In the list of policies, click on your policy, which will open up the two panels below:


Olivia Wagr policy
CLIENT APP POLICY
Delete policy

Essentials

Description

Platform
iOS

Apps
1

All settings

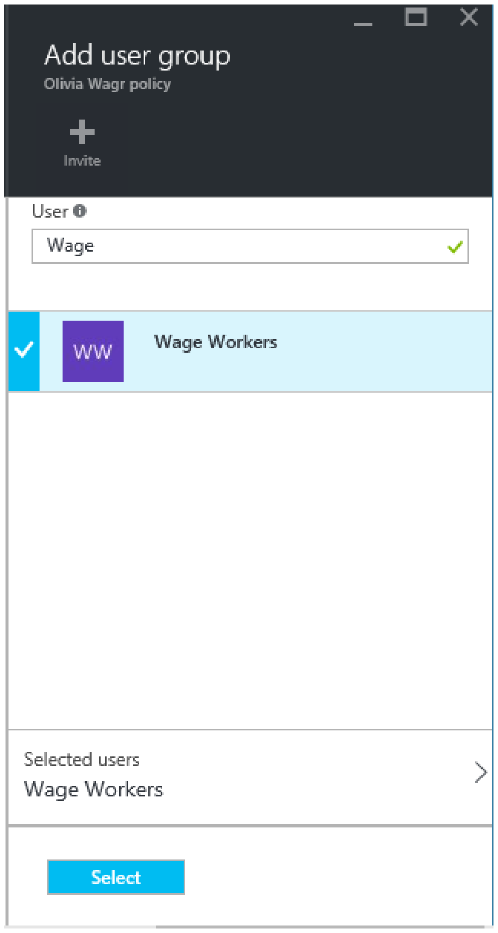
Summary
Add tiles

Olivia Wagr policy
CLIENT APP POLICY
Delete policy

Search to filter settings...

- General
- User groups
- Targeted apps
- Policy settings

Click on "User groups" to select designated target groups. Add a user group to the list of targets by searching by name or selecting from the list of security groups. Click on the Wage Workers group and press select at the bottom of the panel.


Add user group
Olivia Wagr policy
Invite

User
Wage

✓ WW Wage Workers

Selected users
Wage Workers

Select

Congratulations, you've deployed MAM policy. The next time Jane Doe opens Wagr, she'll be asked for her corporate credentials. Then the MAM policy you've enforced as an IT administrator will secure Jane's access to corporate data within the Wagr app.

Deploy the App to End-User Devices

How you get the wrapped versions of your LOB apps onto end-user devices is currently up to you. Without Intune mobile device management (MDM) enrollment, today there is no way to deploy apps onto devices through an Intune channel.

iOS

You can use iTunes or Apple Configurator 2 to sideload the wrapped app onto a test device.

Android

You can directly move the wrapped .apk to the device. From the device, open the .apk to install the app. Make sure the device settings allows for “installation for unknown sources.”

The Intune team is in early stages of development of a tool for App Publishing without Enrollment that allows IT administrators to deploy their LOB apps to devices that are not enrolled with Intune MDM.