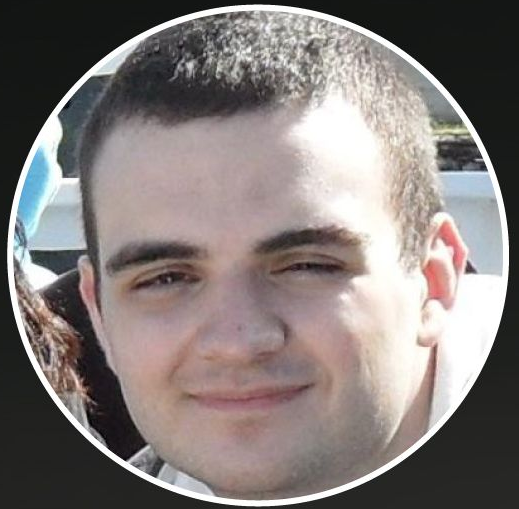
Everything I wish I’d known when creating our mobile development pipeline

J Justin Donaldson
Salesforce

HAXE 2018

US-SUMMIT · SEATTLE · MAY 3-5

FEATURED·SPEAKER



Adam Breece, FlowPlay

LEARN MORE AT
https://summit.haxe.org/us/2018/

flowplay

Example Project

https://github.com/atom-b/hx-summit-2018

@adambreece

Mobile pipeline challenges

- Generated projects add a layer of abstraction
- Platform-specific solutions increase overhead
- iOS tooling is not build automation friendly
- iOS code signing and provisioning are convoluted and docs are terrible

Internal QA Releases

QA Release Requirements

- Available nightly
- On-demand, self-service is highly desirable
- Installable from the device
- Debug symbols

Nightly TestFlight and alpha / beta track builds?

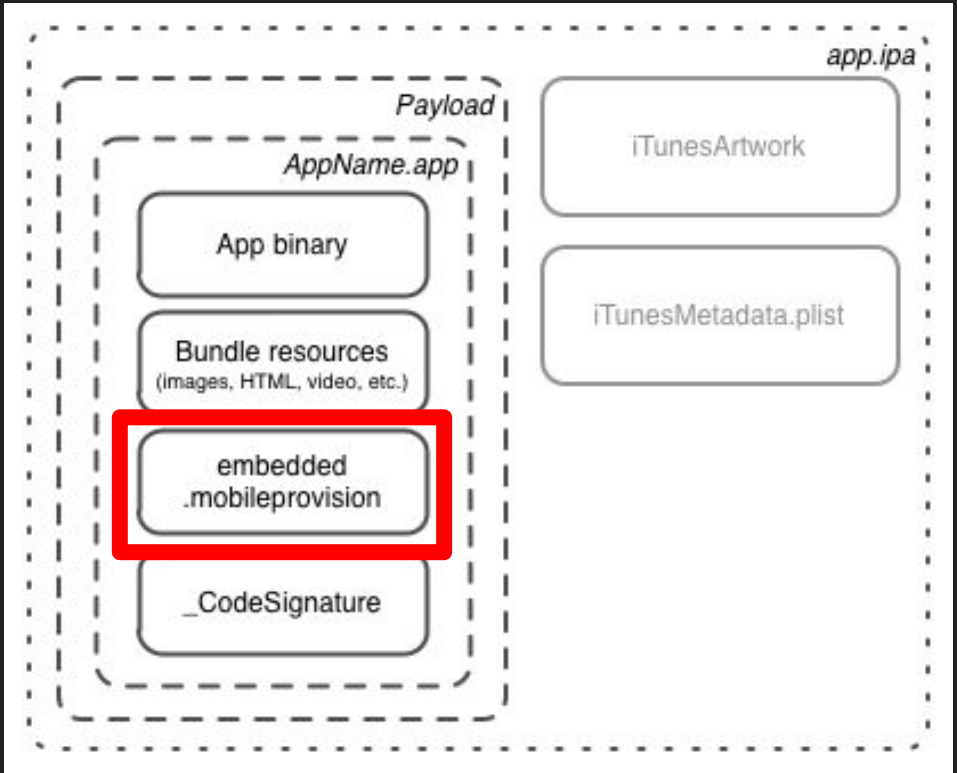
- Often slow to reach devices
- Review policies have blocked our private QA releases
- Slow or no crash log symbolication
- No debug symbol archival / retrieval

Solution: a webpage with the binaries

- Binaries uploaded by CI server
- Android installs APKs directly
- iOS ad-hoc installable via mobile Safari
- Symbols easily accessible, archived

iOS Ad-hoc Distribution

Anatomy of an IPA



https://gowithfloat.com/2011/11/re-signing-an-ios-app-without-xcode/#prettyPhoto/0/

Provisioning Profiles Are Files

- XCode looks for profiles in `~/Library/MobileDevice/Provisioning Profiles/`
- Chooses provisioning profile based on:
  - App ID
  - Development Team ID
  - Deployment method

iOS Provisioning Profile Crash Course

- Deployment method of an IPA
  - Development
  - App Store / Test Flight
  - **Ad-hoc**
  - And more!
- Which devices can install the IPA
  - Developer-approved devices (development, ad-hoc)
  - Any device (App Store, TestFlight)

PiratePig.ipa

Open with iTunes

App info

Name: **Pirate Pig**
Version: 1.0.0 (37)
BundleId: com.flowplay.PiratePig
DeviceFamily: iPhone, iPad

Provisioning

Profile name: **iOS Team Ad Hoc Provisioning Profile: com.flowplay.PiratePig**
Profile UUID: 366f9604-c6a0-456a-8f24-dac682f952c3
Profile Type: iOS Distribution (Ad Hoc)
Team: FlowPlay Inc (AAABBB1234)
Creation date: Apr 20, 2018, 3:46:06 PM (Created today)
Expiration Date: **Jun 28, 2018, 11:33:24 AM (Expires in 68 days)**

DEVELOPER CERTIFICATES

iPhone Distribution: FlowPlay Inc (KAM489HR6S)	Expires

DEVICES (51 DEVICES)

	UDID
0 →	02bd02eb7cc72f2852070349c6c0f3847a3d0fad
	0966d72508e56821e4e55a1416f75616a40fdd8e

Create an IPA with Lime/OpenFL

```
openfl deploy ios [-<deployment method> ...]
```

- Deployment method is any of:
  - `appstore`
  - `adhoc`
  - `enterprise`
  - `development`
- Multiple deployment methods in one command:

```
openfl deploy ios -adhoc -appstore -development
```

- Also generates an XCArchive

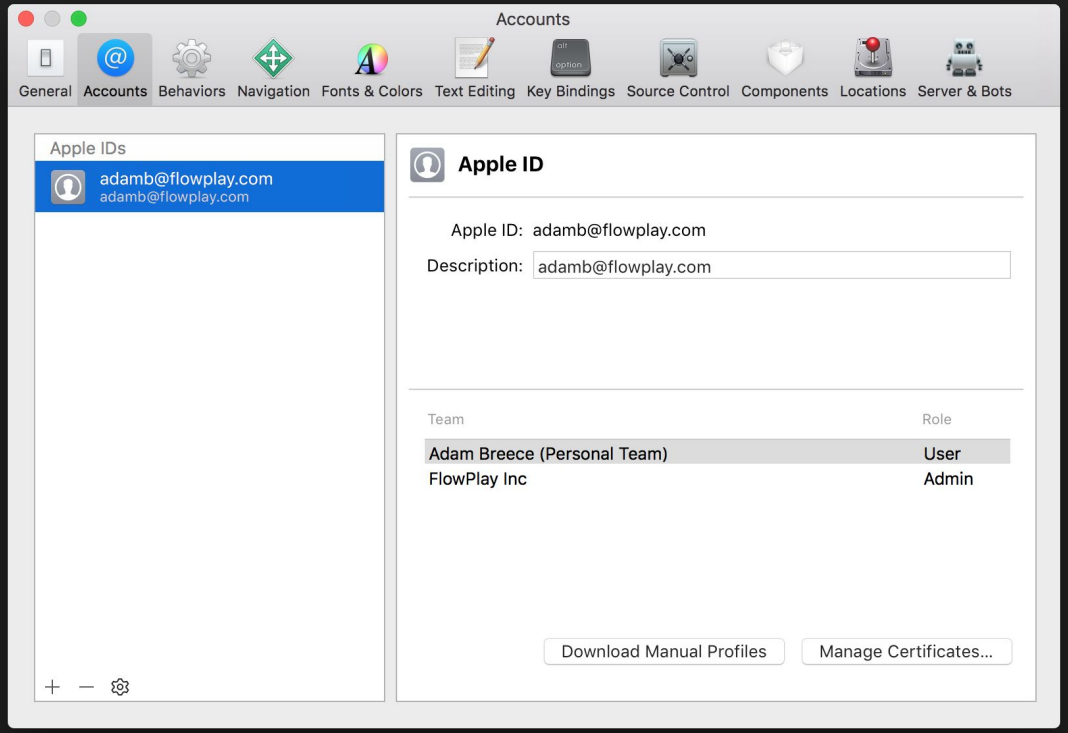
Failing to create an IPA with Lime/OpenFL

```
adam$ openfl deploy ios -adhoc

Code Signing Error: No profiles for 'com.flowplay.PiratePig'
were found:

Xcode couldn't find any iOS App Development provisioning
profiles matching 'com.flowplay.PiratePig'.

Code Signing Error: Code signing is required for product
type 'Application' in SDK 'iOS 11.2'

** ARCHIVE FAILED **
```

Automatic Signing and Provisioning

- Generates a profile for you
- Adds all registered devices to the whitelist for ad-hoc, dev profiles
- New XCode 8, CLI-friendly in XCode 9
- Makes provisioning hands-free (usually)
- Must have an Apple account in XCode
    - On CI servers too!

Add your dev account to XCode



Automatic Provisioning - Project.xml

```
<!-- Automatic provisioning requires a developer account saved in XCode -->
<!-- Tell XCode to update and create provisioning profiles -->


<config:ios allow-provisioning-updates="true"/>
```

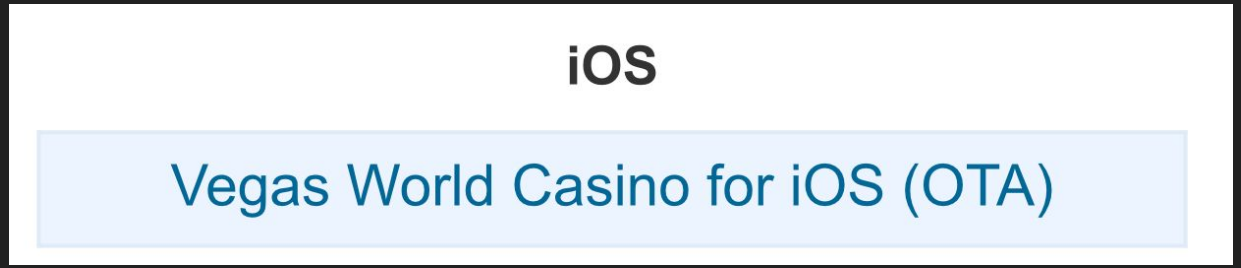
Creating an IPA with Lime/OpenFL (again)

```
adam$ openfl deploy ios -adhoc

Signing Identity:     "iPhone Developer: Adam Breece (ASDF23409W)"
Provisioning Profile: "iOS Team Provisioning Profile: com.flowplay.PiratePig"
                      (9b9fcdde-bf08-4cd9-99f4-f3588327fa9b)

...
** ARCHIVE SUCCEEDED **
...
Exported PiratePig.xcarchive to: ~/piratepig/Export/ios/dist/adhoc
** EXPORT SUCCEEDED **
```

Ad-hoc (OTA) Installs

- Require a `manifest.plist`
- Generated by XCode when exporting ad-hoc IPA
- Safari on iOS uses manifest to perform ad-hoc install
- Devices must click a link to a special URL:

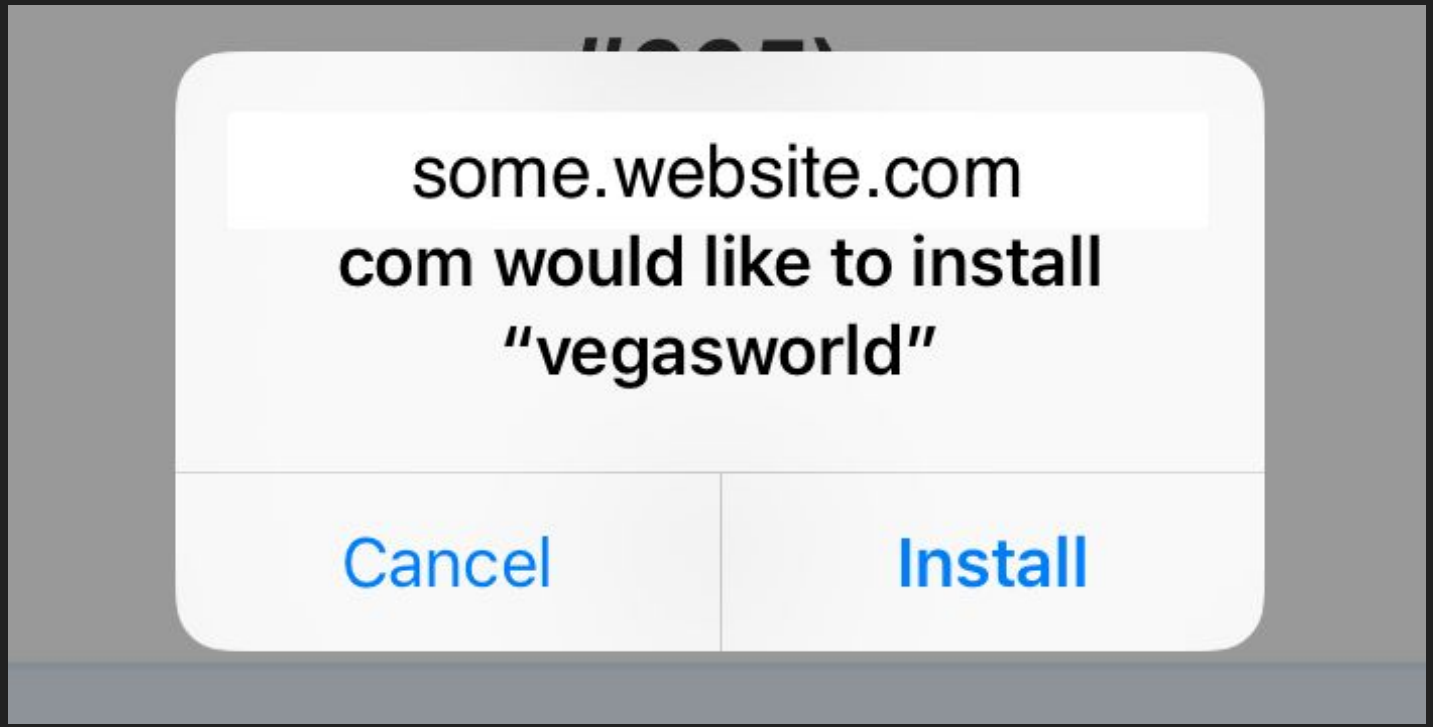
```
itms-services://?action=download-manifest&url=
    "https://some.website.com/dist/manifest.plist"
```



Ad-hoc Manifest Generation - Project.xml

```
<config:ios>
  <adhoc>
    <manifest>
      <appURL>https://some.website.com/dist/PiratePig.ipa</appURL>
      <displayImageURL>https://some.website.com/dist/DISPLAY.png</displayImageURL>
      <fullSizeImageURL>https://some.website.com/dist/FULLSIZE.png</fullSizeImageURL>
    </manifest>
  </adhoc>
</config:ios>
```

Ad-hoc builds with lime/OpenFL



Automatic Signing on CI servers

```
Failed to load credentials for user.name@flowplay.com:
Error Domain=DVTSecErrorDomain Code=-25308

"User interaction is not allowed."
```

Fixing Automatic Signing on CI Servers

Unlock the keychain for every build

```
$ /usr/bin/security unlock-keychain -p <keychain password>
```

Fixing Automatic Signing on CI Servers

- MUST give permissions to signing key via the GUI
- Necessary on first build and IPA export after adding a new signing key

Debug Symbols

Turn this...

```
0  0x0000000100e3b6f0 0x100704000 + 7567088
1  0x0000000100e3b0a4 0x100704000 + 7565476
2  0x0000000100e3b430 0x100704000 + 7566384
3  0x0000000100e5f38c 0x100704000 + 7713676
4  0x0000000100e5f9d0 0x100704000 + 7715280
5  0x0000000100d71a94 0x100704000 + 6740628
```

Into this!

```
0   piratepig::PiratePig_obj::__construct() + 7567088 (PiratePig.cpp:0)
1   DocumentClass_obj::__construct(hx::ObjectPtr<openfl::display::DisplayObjectContainer_obj>) + 7565476 (Docum
2   DocumentClass_obj::__alloc(hx::ImmixAllocator*, hx::ObjectPtr<openfl::display::DisplayObjectContainer_obj>)
3   ApplicationMain_obj::start(hx::ObjectPtr<openfl::display::Stage_obj>) + 7713676 (ApplicationMain.cpp:222)
4   ApplicationMain_obj::create(Dynamic)::_hx_Closure_0::__run() + 7715280 (ApplicationMain.cpp:166)
5   lime::app::_Event_Void_Void_obj::dispatch() + 6740628 (_Event_Void_Void.cpp:124)
```

```
0   piratepig::PiratePig_obj::__construct() + 7567088 (PiratePig.cpp:0)
```

HXCPP_DEBUG_LINK

Add symbols to binary, even in release mode

```
$ openfl build ios -DHXCPP_DEBUG_LINK

 <define name="HXCPP_DEBUG_LINK"/>
```

Symbolicating Android crashes

- From crash log file

```
$ ndk-stack -sym <.so directory> -dump <crash log path>
```

- From logcat

```
$ adb logcat | ndk-stack -sym <.so directory>
```

- Debuggable .so under `bin/app/src/main/jniLibs/<arch>/`

Symbolicating iOS crashes

- GUI
  - Double-click .xcarchive in `build/Release-iphoneos` to register with XCode
  - View device crash logs in XCode
  - Optional - drag .crash file to crash log list
- CLI
  - Apple's `symbolicatecrash` tool
  - Many steps to use - see https://stackoverflow.com/a/43539663/576090
- Symbols in .xcarchive and `.dSYM` package in `build/Release-iphoneos`

Strip symbols from release binaries

- Bloat binary size
- Reverse engineering is easier

Pirate Pig Android Binary	
With symbols	Stripped
138 MB	9.9 MB

Stripping symbols from Android binaries

- Stripped when packed into APK
- Requires Gradle plugin version >= 2.2.3

```
<config:android gradle-plugin="2.2.3" />
```

```
<config:android gradle-version="2.14.1" />
```

- Debuggable .so is in: `bin/app/src/main/jniLibs/<arch>/`

Strip symbols from XCode binaries

- Happens automatically by default!
- dSYMs are included in App Store IPAs so Apple can symbolicate them
- Also in .xcarchive for archival purposes

Automated App Store Releases

App Store Release Requirements

- Fully automated process
- Runnable from CI and dev environments
- Debug symbols archived (already solved!)

Automated App Store Release Challenges

- iOS altool only uploads, won’t release to App Store or TestFlight
- No documented iOS App Store API
- Google Play has an API, but no first party Android tools to use it

App Store release automation

https://fastlane.tools

Fastlane

- Fully automated iOS and Google Play releases from the command line
- Highly customizable
- Open source
- Does lots of other things - see https://fastlane.tools

Fastfile

```
lane :publish do
  deliver(
    submit_for_review: true,
    skip_metadata: false,
    ipa: "dist/appstore/PiratePig.ipa",
  )
end
```

Appfile

```
app_identifier("com.flowplay.PiratePig")

apple_id("my.user@flowplay.com")

itc_team_id("12345678")

team_id("AAABBB1234")
```

Appfile

```
require 'java-properties'


v = JavaProperties.load("version.properties")


ENV['APP_VERSION'] =
    "2.#{v[:'com.version.minor']}.#{v[:'com.version.patch']}"
```

Fastlane + Lime/OpenFL

- Templates generate Fastlane config files at build time
- Fastlane files end up where the Fastlane tools expect them
- Allows running of fastlane command line tools as documented
- Reusable with multiple mobile apps
- Not a requirement - non-default paths can be configured

Fastfile Haxe Template

```
::if (SET_PLATFORM == "ios")::
lane :publish do
   deliver(
     submit_for_review: true,
     skip_metadata: false,
     ipa: "dist/appstore/::APP_FILE::.ipa"
   )
 End ::end::
```

Appfile Haxe Template

```
::if (SET_PLATFORM == "ios")::
app_identifier("::APP_PACKAGE::")
apple_id("my.user@flowplay.com")
itc_team_id("12345678")
team_id("::DEVELOPMENT_TEAM_ID::")
::end::
```

Project.xml

```
<!-- iOS Fastlane config files -->
<template path="tmpl/fastlane/Fastfile" rename="../fastlane/Fastfile"/>
<template path="tmpl/fastlane/Appfile" rename="../fastlane/Appfile"/>
<template path="tmpl/fastlane/Gemfile" rename="../Gemfile"/>

<!-- Android Fastlane config files -->
<template path="tmpl/fastlane/Fastfile" rename="fastlane/Fastfile"/>
<template path="tmpl/fastlane/Appfile" rename="fastlane/Appfile"/>
<template path="tmpl/fastlane/Gemfile" rename="Gemfile"/>
```

```
adam$ cd Export/ios
adam$ fastlane publish
Driving the lane 'ios publish' 🚁
--------------------
--- Step: deliver ---
--------------------
Login to iTunes Connect (my.user@flowplay.com)
Login successful
Making sure the latest version on iTunes Connect matches '1.0.0' from the ipa file...
Successfully set the version to '1.0.0'
Successfully uploaded set of metadata to iTunes Connect
Uploading binary to iTunes Connect
Going to upload updated app to iTunes Connect
This might take a few minutes. Please don't interrupt the script.
iTunes Transporter successfully finished its job
-----------------------------------------------------------------
Successfully uploaded package to iTunes Connect. It might take a few minutes until it's visible online.
-----------------------------------------------------------------
```

Successfully uploaded the new binary to iTunes Connect

Resources

Example project - https://github.com/atom-b/hx-summit-2018

Fastlane - https://fastlane.tools

Provisioning QuickLook Plugin - https://github.com/ealeksandrov/ProvisionQL