Building The iOS Wrapper

Version 1.3.0+

Contents

Introduction

In addition to using the Omnis JavaScript Client in the browser on any computer, tablet or mobile device, you can create standalone apps for iOS that have your JavaScript remote form embedded. These can even operate completely offline (if you have a Serverless Client serial).

To do this, we provide a custom app, or "wrapper", project for iOS. This project allows you to build custom apps, which create a thin layer around a simple Web Viewer which can load your JavaScript remote form. They also allow your form access to much of the device's native functionality, such as contacts, GPS, and camera.

This document describes the steps required in order to create and deploy your own customized wrapper app for iOS. It should provide you with all of the information you need to create your own, self-contained, branded mobile app, and deploy it to users manually or through the App Store.

Setting Up The Build Environment

Install xCode

In order to build iOS apps, you will need to install **xCode 6.1.1 or later**. You can download this (on OS X 10.7 or later) through the **Mac App Store**. You should start by installing this.

Set Up Code Signing Requirements

In order to build an app which will run on an iOS device, you need to code sign your app at build time. The first step in this process is to **sign up for one of Apple's iOS Developer Programs**.

Apple gives you 3 options when signing up for their iOS Developer Program:

- **Free -** This will allow you to download the iOS SDK, and test your applications on the simulator, but you cannot distribute your app to a real device.
- **Standard** ($99 per year) - This allows you to submit your app to the AppStore, and also allows you to create and distribute Ad-Hoc apps to run on up to 100 specified devices.
- **Enterprise** ($299 per year) - This option is for larger corporations only. Your company must have a Dun & Broadstreet Number (DUNS). This does not allow the distribution of Apps through the AppStore, but allows distribution to a greater number of (unspecified) in-house devices.

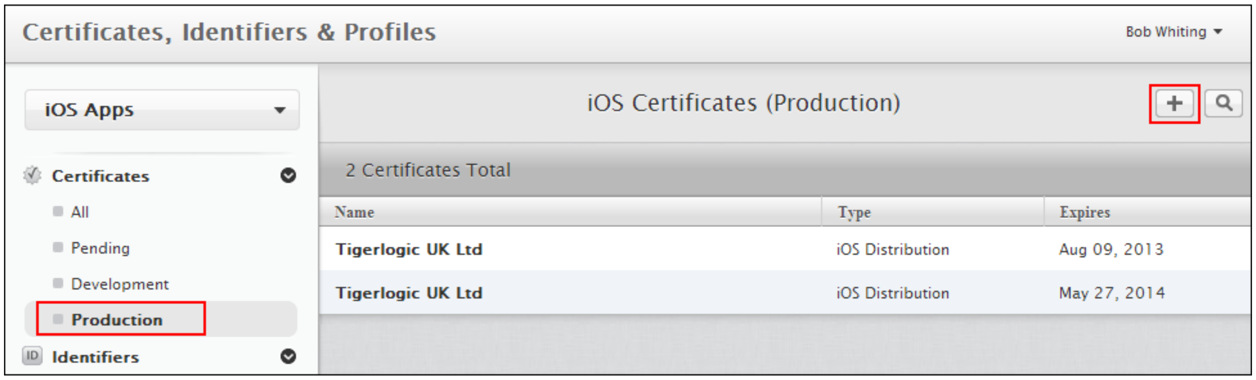
We expect the great majority of Omnis developers to sign up for the **Standard** program, and this is used for the basis of the tech note.
You are also given the choice of signing up as a Company, or as an Individual. Signing up as a Company gives you the ability to add team members, whereas signing up as an Individual does not.

Once you have signed up as an iOS developer, you should **sign in to the iOS Dev Center**, then follow through the steps below:

Certificates

- Open the **Certificates** section of your iOS Dev Center account, and select **Production** certificates.

- Push the **+** button to open a wizard to take you through the creation of a new certificate.

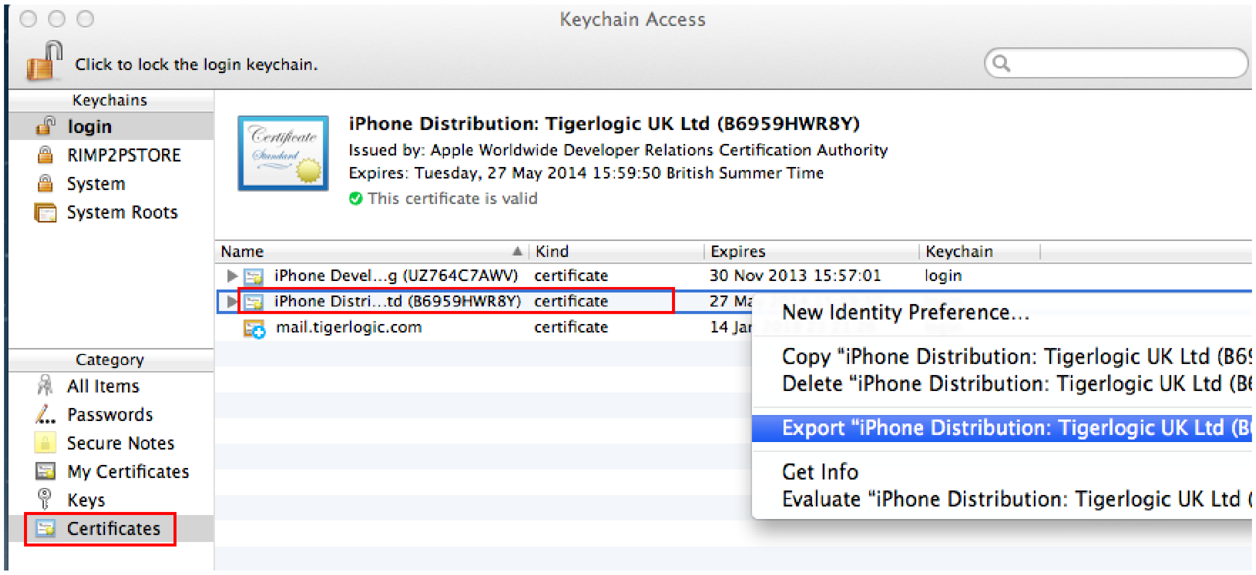


Name	Type	Expires
Tigerlogic UK Ltd	iOS Distribution	Aug 09, 2013
Tigerlogic UK Ltd	iOS Distribution	May 27, 2014

- When prompted to select the type of certificate, you should choose an **App Store and Ad Hoc Production** certificate.
    - You should also use the link provided on this page to download and install the **Intermediate Certificates** *(Worldwide Developer Relations Certificate Authority)*, if you do not already have it installed.

- The wizard will then guide you through the rest of the process to create your certificate.

- Once you have created your certificate (and its associated private key), it is important that you **BACK THIS UP**.
    - Open **Keychain Access** from your Mac's *Applications/Utilities/*.
    - Select the **Certificates** Category from the sidebar, and locate your certificate you just created.
    - Right click the certificate and select **Export**.
    - Keep this somewhere safe - if you change machines, or for any reason lose the certificate from your keychain, you can import the certificate from this backup. If you want to build an update to any of your apps, it must be signed with the same certificate, so this is important.


Keychain Access
Click to lock the login keychain.
Keychains
login
RIMP2PSTORE
System
System Roots
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
New Identity Preference…
Copy "iPhone Distribution: Tigerlogic UK Ltd (B6
Delete "iPhone Distribution: Tigerlogic UK Ltd (B
Export "iPhone Distribution: Tigerlogic UK Ltd (B
Get Info
Evaluate "iPhone Distribution: Tigerlogic UK Ltd (

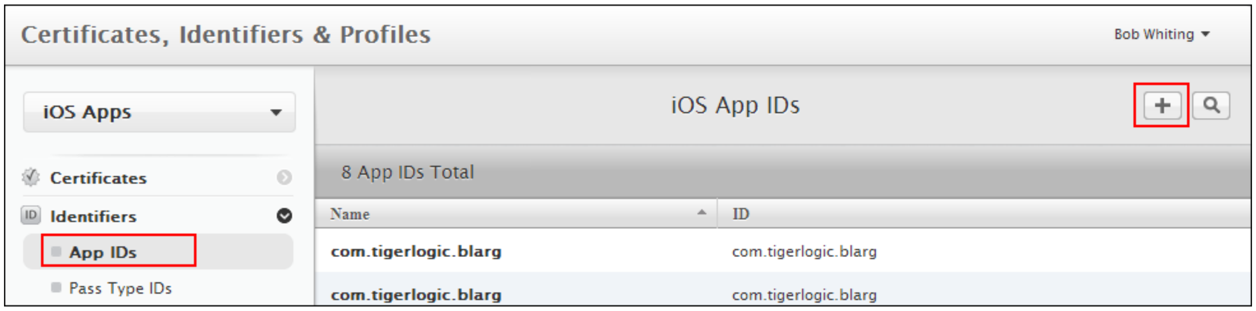
iPhone Distribution: Tigerlogic UK Ltd (B6959HWR8Y)
Issued by: Apple Worldwide Developer Relations Certification Authority
Expires: Tuesday, 27 May 2014 15:59:50 British Summer Time
This certificate is valid

Name	Kind	Expires	Keychain
iPhone Devel...g (UZ764C7AWV)	certificate	30 Nov 2013 15:57:01	login
iPhone Distri...td (B6959HWR8Y)	certificate	27 Ma	
mail.tigerlogic.com	certificate	14 Jar	

Identifiers

An App ID determines which app **Identifiers** you will be able to sign with the profile you are creating.

- Open the **Identifiers** section of your iOS Dev Center account, and select **App IDs**.
- Push the **+** button to open a wizard to take you through the creation of a new App ID.



8 App IDs Total

Name	ID
com.tigerlogic.blarg	com.tigerlogic.blarg
com.tigerlogic.blarg	com.tigerlogic.blarg

- You can choose to create an Explicit App ID (can only be used to sign a single app identifier), or a Wildcard App ID (can be used to sign any app whose Identifier matches the pattern specified).
- The convention for app Identifiers is to use a reverse domain name. E.g.**com.mycompany.myapp**. So you could set this as an explicit App ID, or if you are using a wildcard App ID you could use anything from "*****" to"**com.mycompany.***".

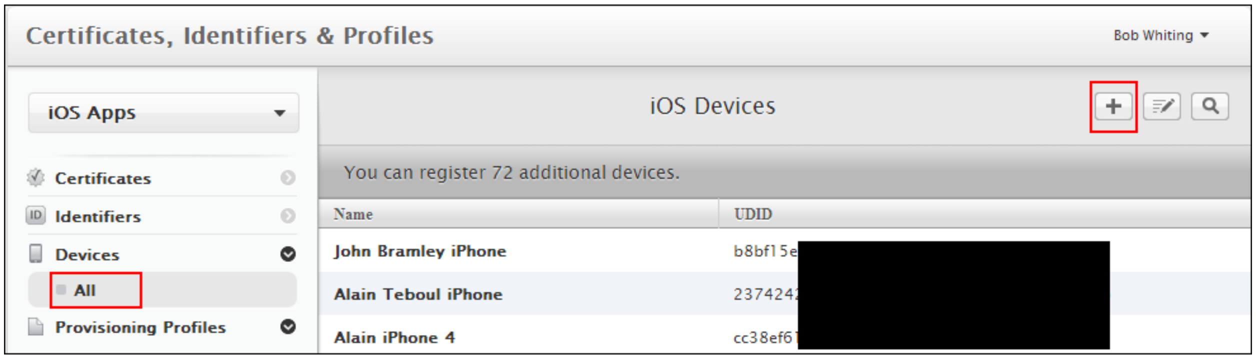
*Whichever you choose, **make a note of this**, as it will be needed later when you assign an **Identifier** to your app.*

- Omnis wrapper apps do not require any *App Services*.

Devices

If you are going to deploy via the Ad-Hoc method (not through the App Store), you need to explicitly define each individual device which your app will run on.

- Open the **Devices** section of your iOS Dev Center account.
- Push the **+** button to open a page to allow you to add a new device.



Certificates, Identifiers & Profiles — Bob Whiting

iOS Devices

You can register 72 additional devices.

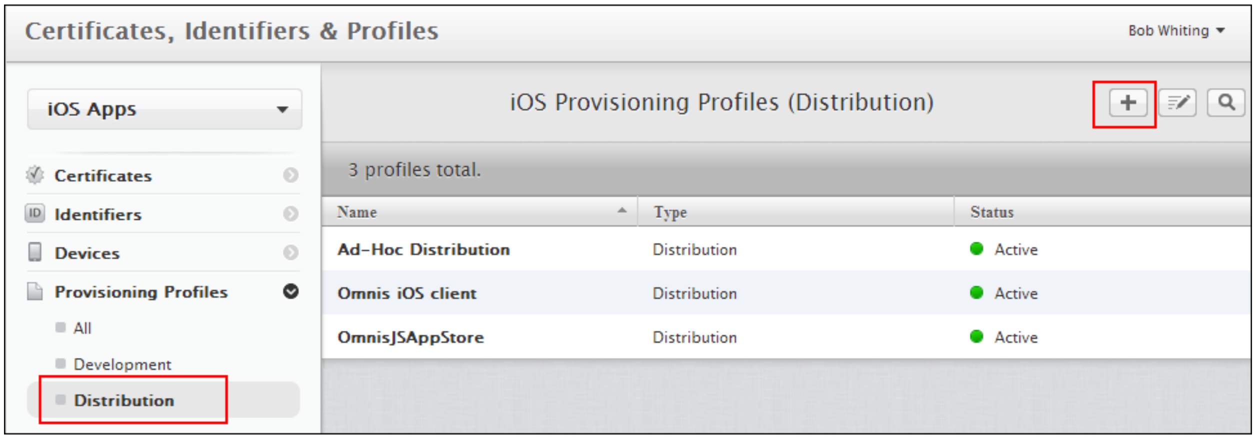
Name	UDID
John Bramley iPhone	b8bf15e[redacted]
Alain Teboul iPhone	237424[redacted]
Alain iPhone 4	cc38ef6[redacted]

- Devices are added by their **UDID** (Unique Device Identifier). You can find your device's UDID by connecting it to iTunes, and clicking on its serial.
  - **whatsmyudid.com** provides a simple tutorial for this.

Provisioning Profile

A Provisioning Profile ties together a Certificate and an App ID (and a group of Devices for Ad Hoc Provisioning Profiles). It is with this that you then sign your app (in association with the matched certificate/private key pair stored in your keychain).

- Open the **Provisioning Profiles** section of your iOS Dev Center account, and select **Distribution**.
- Push the **+** button to open a wizard to take you through the creation of a new Provisioning Profile.



Certificates, Identifiers & Profiles — Bob Whiting

iOS Provisioning Profiles (Distribution)

3 profiles total.

Name	Type	Status
Ad-Hoc Distribution	Distribution	● Active
Omnis iOS client	Distribution	● Active
OmnisJSAppStore	Distribution	● Active

- You will be offered the choice of creating an **Ad Hoc** or **App store** Distribution Provisioning Profile. You should select that which matches the distribution model you are going to use. Make sure you select a **Distribution** profile - not Development.

It is possible to create multiple Provisioning Profiles, so you can create one (or more) of each type, should you wish.

- Follow through the wizard, and once complete, you will be able to download the provisioning profile. Do so, then double-click the downloaded file in the Finder, and it will be imported into xCode.

Open The Project

- First, download the latest iOS wrapper project **from our website**.
- Extract the contents of the wrapper zip file to a folder on your Mac. Make sure that there are no spaces in the path to the extracted folder.
- Double-click the **OmnisJSWrapper.xcodeproj** file, to open the project in xCode.

Configuring The App

Configuration of the app is done through the **config.xml** file, which is situated in the root of your project. You should set the values in this config file to point the app to your Omnis server, and to configure how the app behaves.

The properties within the config file are as follows:

Behavior Settings (UPDATED)

- **AppTitle**: Whether the app should show the status bar at the top. (1 for yes, 0 for no).
- **LightStatusBar**: Whether the status bar should use **white** (1) or **black** (0) text. Only applies if *AppTitle* is enabled.
- **MenuIncludeOffline**: Whether the pull-down menu includes the option to switch between online & offline modes. (1 for yes, 0 for no).
- **MenuIncludeUpdate**: Whether the pull-down menu includes the option to update when in offline mode. (1 for yes, 0 for no).
- **MenuIncludeAbout (NEW)**: Whether the pull-down menu includes the option to show the About screen. (1 for yes, 0 for no).

*The **About** screen contains a link to the **Credits** screen. MenuIncludeAbout is set to 0, the **Credits** option will instead be shown in the pull-down menu.*

- **SettingsFloatControls**: Whether controls on the form should float (using their designed $edgefloat property) to adapt to the device's full screen size. Does not apply if *SettingsScaleForm* is enabled. (1 for yes, 0 for no). **Recommended set to 1**.

- **SettingsScaleForm**: Whether the designed form should be scaled to fit the current device's screen. (1 for yes, 0 for no). **Recommended set to 0.**

- **SettingsAllowHScroll & SettingsAllowVScroll**: Set these to 1 if you want to allow horizontal or vertical scrolling of the form respectively, or 0 if not.

- **SettingsMaintainAspectRatio**: If *SettingsScaleForm* is set to 1, this controls whether the scaling maintains the design form's aspect ratio. 1 for yes, 0 for no.

- **SettingsOnlineMode**: Whether the app should initially start in online mode (set to 1), or offline mode (set to 0).

- **ServerLocalDatabaseName**: The name (including .db extension) of the local sqlite database to use. If you are bundling a prepopulated database with your app, its name should match that which you set here.

- **UseLocalTime**: If 0, dates & times are converted to/from UTC, as default. Setting this to 1 will disable this conversion. (Offline only - online mode reads from remote task's *$localtime* property).

Connection Settings

- **ServerOmnisWebUrl**: URL to the Omnis or Web Server. If using the Omnis Server it should be *http://<ipaddress>:<omnis port>*. If using a web server it should be a URL to the root of your Web server. E.g. *http://myserver.com*

- **ServerOnlineFormName**: Route to the form's .htm file from *ServerOmnisWebUrl*. So if you're using the built in Omnis server, it will be of the form **/jschtml/myform**. If you are using a web server, it will be the remainder of the URL to get to the form, e.g. **/omnisapps/myform**. (**Do not add the .htm extension!**)

*Only **ServerOmnisWebUrl** & **ServerOnlineFormName** are needed for Online forms. The following Server… properties are needed in addition to ServerOmnisWebUrl for Offline mode only.*

- **ServerOmnisServer**: The Omnis Server *<IP Address>:<Port>*. Only necessary if you are using a web server with the Omnis Web Server Plugin. If the Omnis App Server is running on the same machine as the web server, you can just supply a port here.
  E.g. 194.168.1.49:5912

- **ServerOmnisPlugin**: If you are using a web server plug-in to talk to Omnis, the route to this from *ServerOmnisWebUrl*.
  E.g. /cgi-bin/omnisapi.dll

- **ServerOfflineFormName**: Name of the offline form. (**Do not add .htm extension!**)

- **ServerAppScafName**: Name of the App SCAF. This will be the same as your library name.

Note: this is case-sensitive and must match the App Scaf (by default this is generally all lower-case).

- **TestModeEnabled**: Enable test mode (Ctrl-M on form from Studio to test on device).
  Make sure to disable before publishing your release app. (1 to enable, 0 to disable)

- **TestModeServerAndPort**: The *<ipaddress>:<port>* of the Omnis Studio Dev version you wish to use test mode with.

NOTE: The values in the config.xml file are currently read only on first launch of the app. They are then saved to, and read from, local storage to improve performance and allow in-app configuration from the Settings screen.
As such, if you make changes to the config.xml, you will need to uninstall the old app from your device before running the new version.

Customizing The App

Once you have imported the wrapper project into xCode, you should customize it for your particular application.

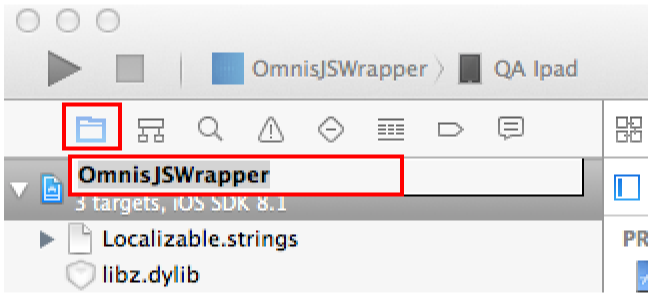
Rename The Project (UPDATED)

NOTE: There seems to be an issue in Xcode 6.3 whereby it will crash on renaming the project.
If you are using Xcode 6.3 you should not attempt to rename your project.

Once you have opened the wrapper project in xCode, you will probably want to give it a name which reflects your particular application.
Note that naming the project has no effect on the resulting app, but just better allows you to keep track of your projects, especially if you are creating multiple different apps. You should use a separate project for each app you create.

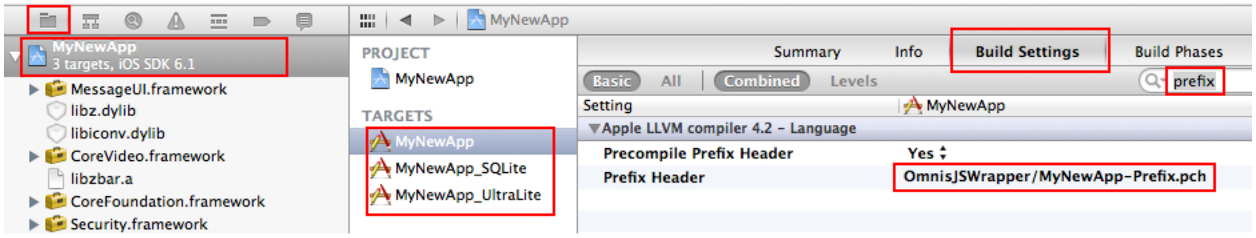
- Show the **Project Navigator** view in the project's sidebar (click the folder icon in the sidebar's toolbar).

- Select the project name at the top level of this view, then push **Enter** to allow you to rename the project.



- Change the name, then push **Enter** again. You will be prompted as to whether you'd also like to rename various other instances - select to **rename all** of these.

- The current version of xCode (at the time of writing) seems to miss renaming the Prefix Header of other targets. So you need to manually change this:
    - Select the root level of your project in the Project Navigator - this will bring up your project settings in the main pane.
    - Select one of the **Targets**, and view its **Build Settings**.
    - Locate the **Prefix Header** setting (you may like to make use of the *search* box), and ensure that the *.pch* file name is set to **<NewProjectName>-Prefix.pch**.
    - Repeat this for each of the 3 Targets.



Precompile Prefix Header: Yes
Prefix Header: OmnisJSWrapper/MyNewApp-Prefix.pch

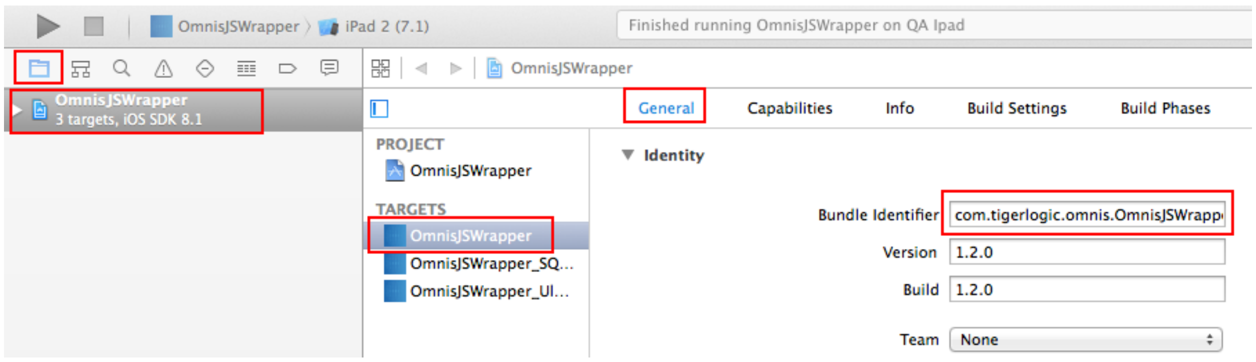
Change The Identifier

The **Identifier** identifies your app, and must be unique amongst all apps on the device. Two apps with the same Identifier will be seen by the device as the same app, so this is an important step.
As such, it is recommended to use a reverse domain name syntax. E.g *com.mycompany.omnis.myfirstapp*.

- Select the root of your project in the *Project Navigator.*

- Select any of the *Targets*, and open its **General** tab.

- Change the **Bundle Identifier** to your own value.



Bundle Identifier com.tigerlogic.omnis.OmnisJSWrapp

Version 1.2.0

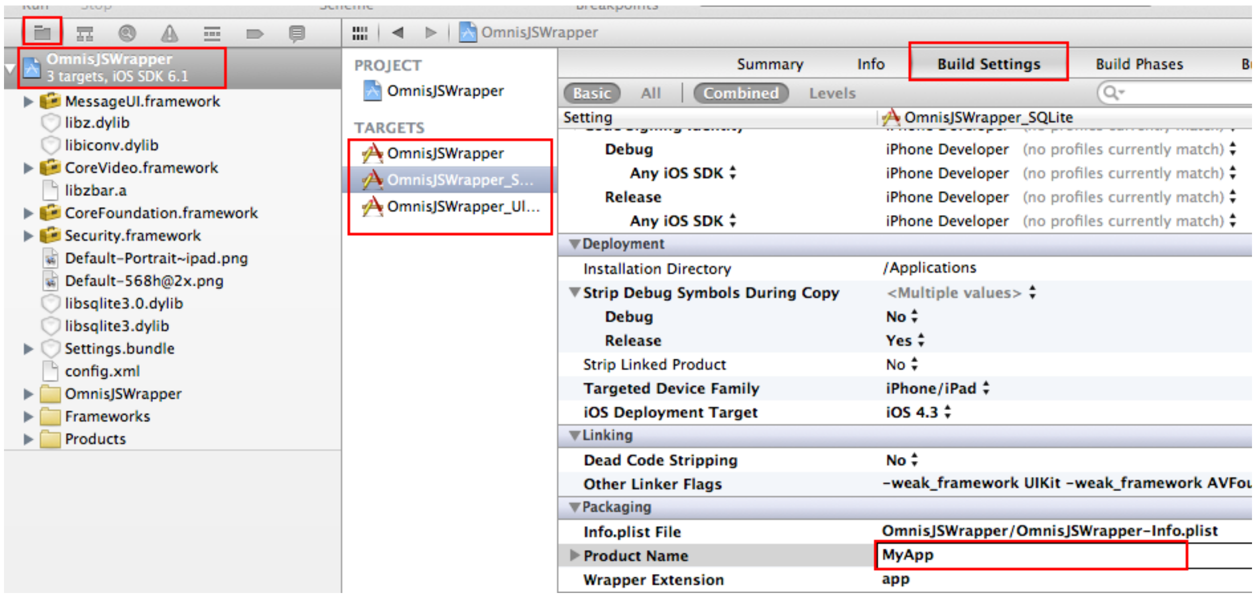
Build 1.2.0

Team None

Change The App Name

To change the display name of your app:

- Select the root of your project in the *Project Navigator*.
- Select one of the *Targets*, and open its **Build Settings** tab.
- Locate the **Product Name** in the list of settings, and change it to your own value.
- Repeat this for each of the targets.



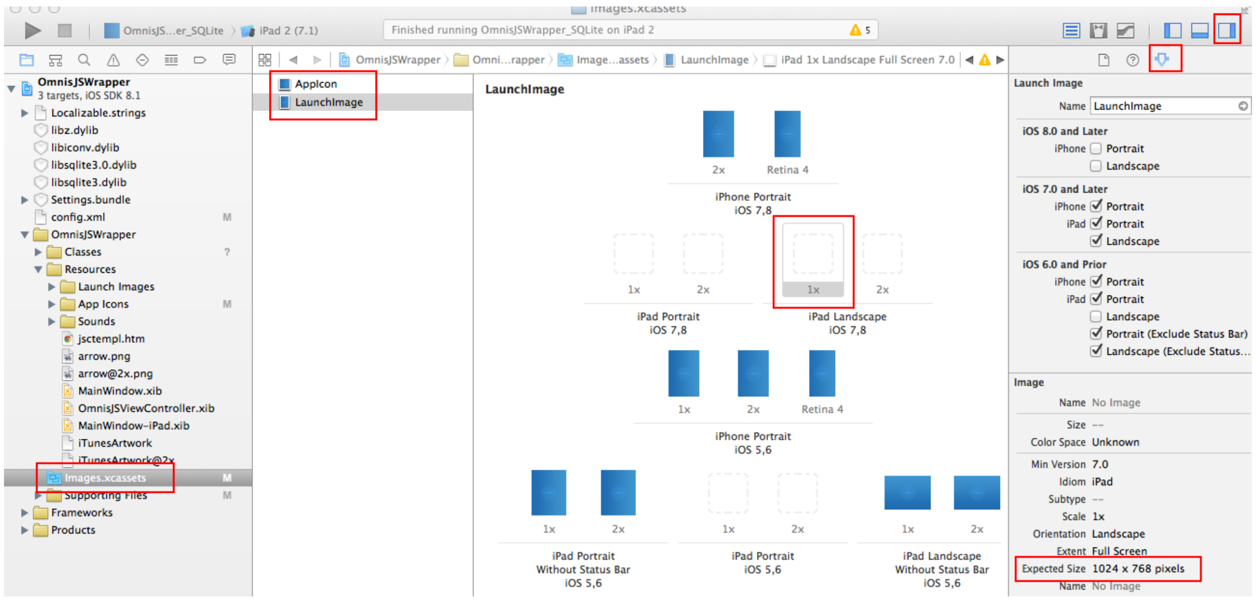
Setting	OmnisJSWrapper_SQLite
Debug	iPhone Developer (no profiles currently match)
Any iOS SDK	iPhone Developer (no profiles currently match)
Release	iPhone Developer (no profiles currently match)
Any iOS SDK	iPhone Developer (no profiles currently match)
Deployment	
Installation Directory	/Applications
Strip Debug Symbols During Copy	<Multiple values>
Debug	No
Release	Yes
Strip Linked Product	No
Targeted Device Family	iPhone/iPad
iOS Deployment Target	iOS 4.3
Linking	
Dead Code Stripping	No
Other Linker Flags	-weak_framework UIKit -weak_framework AVFou
Packaging	
Info.plist File	OmnisJSWrapper/OmnisJSWrapper-Info.plist
Product Name	MyApp
Wrapper Extension	app

Add Custom Icons & Splash Screens

The images used for the Icons and splash screens are stored in an *asset catalog* named **Images.xcassets** in the project's **OmnisJSWrapper** folder. You should replace these with your own versions.

- Select **Images.xcassets** in the *Project Navigator* (in the *OmnisJSWrapper* folder), to view it in the asset catalog editor.

- Select **AppIcon** in the sidebar to view the catalog of app icons.
    - Here you can see your current icons for each specified size. These are all square icons, so e.g. 40pt means an image source sized 40x40.
    - To add/update an icon, you just need to drag the appropriately sized source image from the Finder, or your Project, onto the appropriate space.



Launch Image
Name LaunchImage

iOS 8.0 and Later
iPhone ☐ Portrait
☐ Landscape

iOS 7.0 and Later
iPhone ✓ Portrait
iPad ✓ Portrait
✓ Landscape

iOS 6.0 and Prior
iPhone ✓ Portrait
iPad ✓ Portrait
☐ Landscape
✓ Portrait (Exclude Status Bar)
✓ Landscape (Exclude Status...

Image
Name No Image
Size --
Color Space Unknown
Min Version 7.0
Idiom iPad
Subtype --
Scale 1x
Orientation Landscape
Extent Full Screen
Expected Size 1024 x 768 pixels
Name No Image

- Select **LaunchImage** in the sidebar to view the catalog of launch images (similar to a splash screen, but should be designed to match the background of your app, to give the appearance that your app is starting more quickly).
    - As before, you can update the images by dragging your own image files into the appropriate spaces.
    - In order to find the appropriate size for a particular image, select its space in the asset catalog editor, and in the *Utilities* pane on the right, select the **Attributes Inspector** view , and you will see the **Expected Size** displayed.
        - If the right-hand pane is not shown, click the **Show or Hide Utilities** button on the far right end of the toolbar .

NOTE: The wrapper project does not ship with image sizes to cover all devices and versions. The supplied images will however work with all iOS devices & versions (from iOS 6).
However, for the best end-user experience (to avoid any icons being scaled) we recommend you create images to match each of the sizes specified by the asset catalog editor.

App Icons For iOS < 5

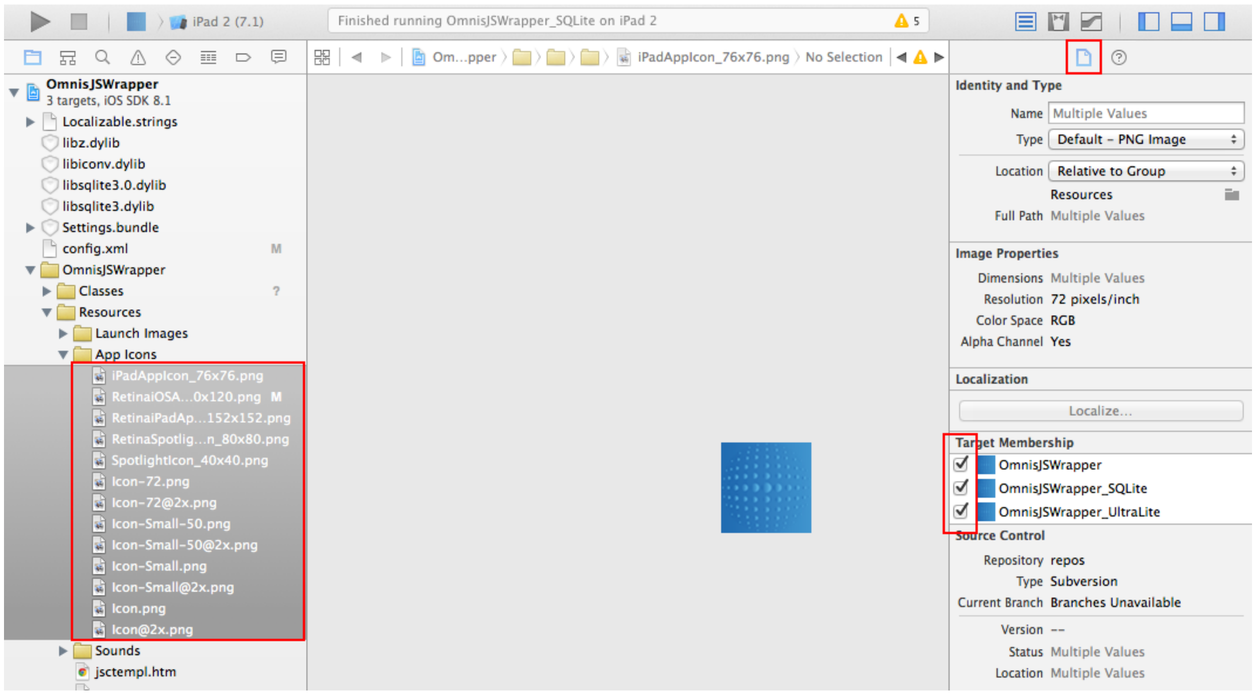
iOS devices running iOS prior to iOS 5 do not understand App Icons in the asset catalog image format, and using the above technique may result in blank icons for these devices.

If you wish to support these devices, you will need to include some duplicate images in your app.

- Using the *Project Navigator*, drill down to **OmnisJSWrapper/App Icons**. In this folder are a collection of app images which will be used for devices which don't support Asset Catalog icons.

- You should replace these files with your own images, making sure to keep the names and sizes the same.

- By default, to keep the size of the app down, these images are not included in the compiled app. In order to add them, you need to:
  - Select all of the image files in the *Project Navigator*.
  - Open the **File Inspector** in the **Utilities** pane (on the right).
  - Under **Target Membership**, tick the box next to all of the targets. This means that the files will be included when compiling that target.



Identity and Type
Name Multiple Values
Type Default – PNG Image
Location Relative to Group
Resources
Full Path Multiple Values

Image Properties
Dimensions Multiple Values
Resolution 72 pixels/inch
Color Space RGB
Alpha Channel Yes

Localization

Target Membership
☑ OmnisJSWrapper
☑ OmnisJSWrapper_SQLite
☑ OmnisJSWrapper_UltraLite

Source Control
Repository repos
Type Subversion
Current Branch Branches Unavailable
Version --
Status Multiple Values
Location Multiple Values

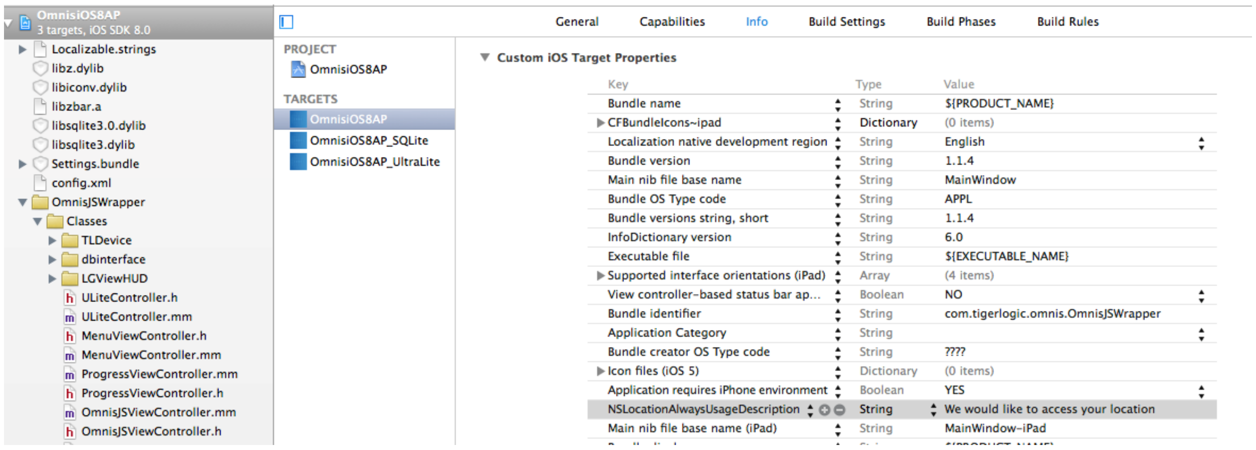
Requesting GPS

When GPS is required in your app, via the Device control, the end user receives a prompt to agree to the use of GPS. Apple changed the APIs for this in iOS8 and included a new requirement for all apps through the app store. You need to make sure one of two strings exists in your iOS project, either using the default string, or adding your own. This string is shown on the prompt on the end user's device when requesting access to GPS.

The default string **NSLocationAlwaysUsageDescription** which allows the app to access GPS in the background and foreground. You can change the text of this to something more appropriate for your app.

You can remove this string and add another string **NSLocationWhenInUseUsageDescription** (also with a description). This will limit the app to GPS data ONLY when running and active with no background support.

Only one of these strings should be included in the app.


OmnisiOS8AP
3 targets, iOS SDK 8.0

- Localizable.strings
- libz.dylib
- libiconv.dylib
- libzbar.a
- libsqlite3.0.dylib
- libsqlite3.dylib
- Settings.bundle
- config.xml
- OmnisJSWrapper
  - Classes
    - TLDevice
    - dbinterface
    - LGViewHUD
    - ULiteController.h
    - ULiteController.mm
    - MenuViewController.h
    - MenuViewController.mm
    - ProgressViewController.mm
    - ProgressViewController.h
    - OmnisJSViewController.mm
    - OmnisJSViewController.h

PROJECT
OmnisiOS8AP

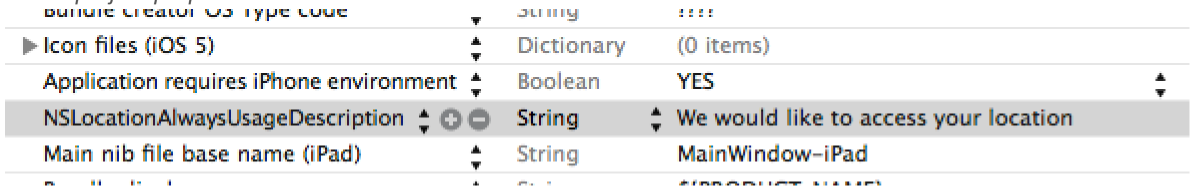
TARGETS
OmnisiOS8AP
OmnisiOS8AP_SQLite
OmnisiOS8AP_UltraLite

General | Capabilities | Info | Build Settings | Build Phases | Build Rules

Custom iOS Target Properties

Key	Type	Value
Bundle name	String	${PRODUCT_NAME}
CFBundleIcons~ipad	Dictionary	(0 items)
Localization native development region	String	English
Bundle version	String	1.1.4
Main nib file base name	String	MainWindow
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.1.4
InfoDictionary version	String	6.0
Executable file	String	${EXECUTABLE_NAME}
Supported interface orientations (iPad)	Array	(4 items)
View controller-based status bar ap...	Boolean	NO
Bundle identifier	String	com.tigerlogic.omnis.OmnisJSWrapper
Application Category	String	
Bundle creator OS Type code	String	????
Icon files (iOS 5)	Dictionary	(0 items)
Application requires iPhone environment	Boolean	YES
NSLocationAlwaysUsageDescription	String	We would like to access your location
Main nib file base name (iPad)	String	MainWindow~iPad

Detail of project properties.

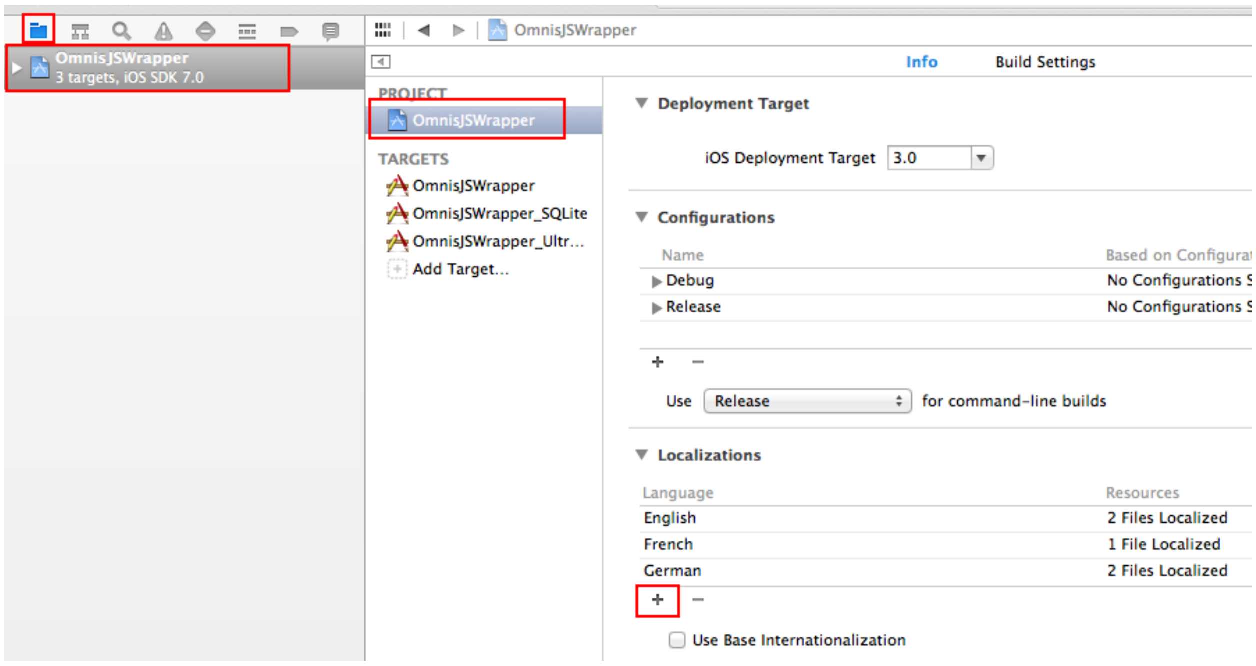


Icon files (iOS 5)	Dictionary	(0 items)
Application requires iPhone environment	Boolean	YES
NSLocationAlwaysUsageDescription	String	We would like to access your location
Main nib file base name (iPad)	String	MainWindow~iPad

Localize App

If you wish to translate text used by the wrapper app, you can do so as described here. If the user's device is set to one of the supported languages, it will use the specified translated strings.

- Select the top level of the project in the *Project Navigator*, and select the project's **Info** pane.
- Locate the **Localizations** section, and add a new language.



iOS Deployment Target 3.0

Name	Based on Configurat
Debug	No Configurations S
Release	No Configurations S

Use Release for command-line builds

Language	Resources
English	2 Files Localized
French	1 File Localized
German	2 Files Localized

Use Base Internationalization

- In the window which appears, make sure you select both the **Localizable.strings** and **InfoPlist.strings** files to be localized.

- Now if you locate **Localizable.strings** in the Project Navigator, you will see that it can be expanded. If you do so, you will see that a copy of the file has been created for your new language.

- This file contains a group of key-value pairs. The **keys** (to the left of the equals signs) must be left as they are. The **values** (to the right of the equals signs) should be translated to your new language.

- Make sure to **always** end each line with a semi-colon.

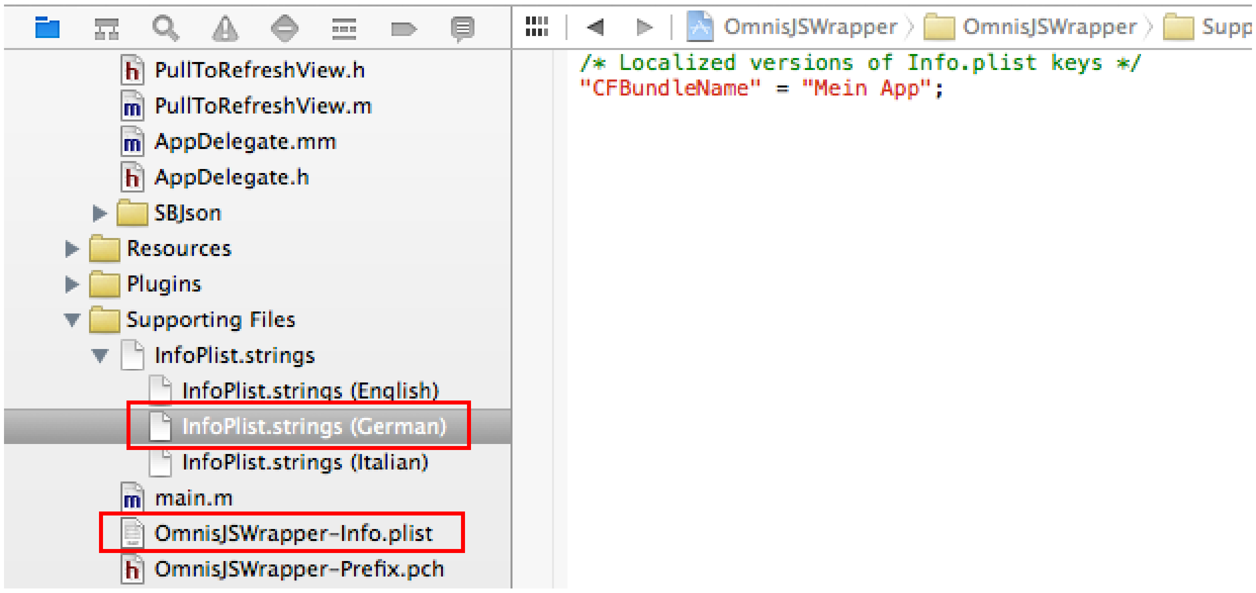
Similarly, if you wish to change some of the attributes of your application package (e.g. The App name, icon etc), you can do so by:

- Locate the **InfoPlist.strings** file in your project, under **Supporting Files**, expand this and select the file which corresponds to the appropriate language.

- This file can be built up using key-value pairs, in the same way that the **Localizable.strings** files are. i.e: "Key Name" = "Key Value";

- The key names in this case must match a key defined in your **<Project Name>-Info.plist** file. These keys aren't displayed with the default Property List viewer, so right click the file and **Open As > Source Code** to see these.

- Remember to end each line with a **semi-colon**.



```
/* Localized versions of Info.plist keys */
"CFBundleName" = "Mein App";
```

Edit The About screen (NEW)

If enabled in your app's **config.xml**, the About screen can be accessed from the pull-down menu.

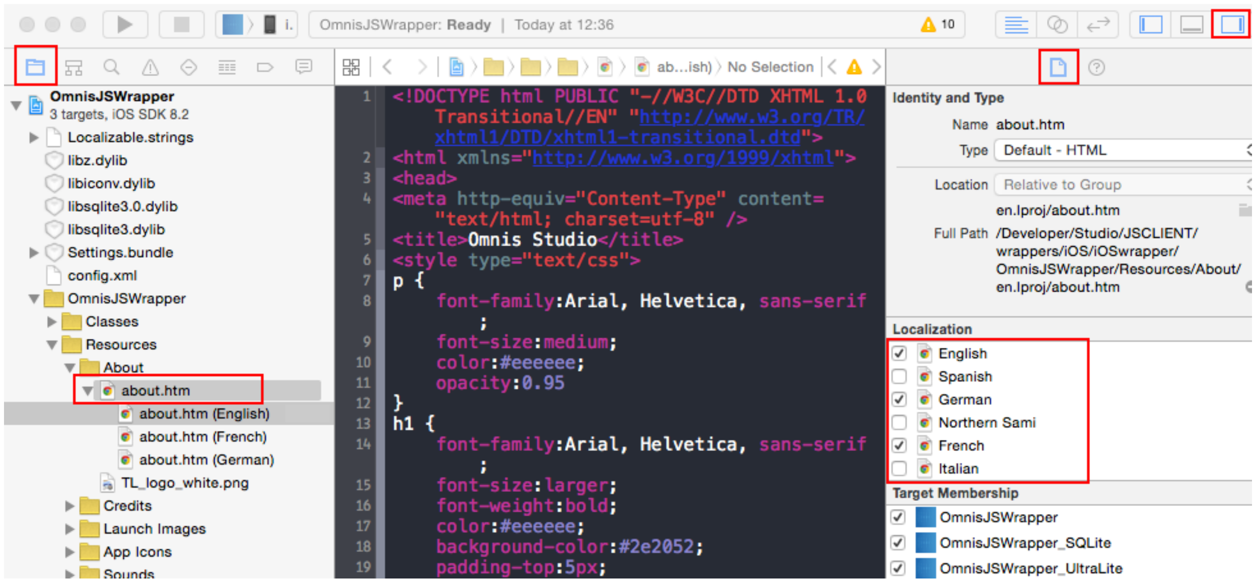
You will want to customize the default About page to reflect your app/brand.
The About page is formatted as html, to enable you to easily customize its content.

- Browse to the **Resources/About** folder in the Project Navigator.
- This folder contains **about.htm**, which is the content of the About page. You should customize this as you see fit for your application.
- Any local resources your page may need can also be added to this folder, as demonstrated by the *TL_logo_white.png* image used by the default About page.

Localize The About Screen

If you are localizing your app, you will probably want to provide a translated About page for your supported languages.
Once you've set up localization, as described in the Localize App section, this is very straightforward:

- Select the **about.htm** file in the Project Navigator.
- In the **Utilities** pane (the left-hand pane of Xcode), show the **file Inspector** view, and select the languages you wish to localize for in the **Localization** section.
- This will add a drop-arrow to **about.htm** in the Project Navigator - if you drop this, you can edit a separate file for each language.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Transitional//EN" "http://www.w3.org/TR/
    xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content=
    "text/html; charset=utf-8" />
<title>Omnis Studio</title>
<style type="text/css">
p {
    font-family:Arial, Helvetica, sans-serif
        ;
    font-size:medium;
    color:#eeeeee;
    opacity:0.95
}
h1 {
    font-family:Arial, Helvetica, sans-serif
        ;
    font-size:larger;
    font-weight:bold;
    color:#eeeeee;
    background-color:#2e2052;
    padding-top:5px;
```

Identity and Type

Name about.htm
Type Default - HTML
Location Relative to Group
en.lproj/about.htm
Full Path /Developer/Studio/JSCLIENT/wrappers/iOS/iOSwrapper/OmnisJSWrapper/Resources/About/en.lproj/about.htm

Localization

- [✓] English
- [ ] Spanish
- [✓] German
- [ ] Northern Sami
- [✓] French
- [ ] Italian

Target Membership

- [✓] OmnisJSWrapper
- [✓] OmnisJSWrapper_SQLite
- [✓] OmnisJSWrapper_UltraLite

Edit The Credits Screen (NEW)

If the **About** menu option is enabled in the config.xml, the About screen will have a link to the Credits screen in its Navigation Bar. Otherwise, a **Credits** option is displayed in your app's pull-down menu.

NOTE: The **Credits** page **MUST** be accessible from your app.

The **Credits** screen works in a similar way to the **About** screen - displaying the contents of the **credits.html** file from your project's **Resources/Credits** folder. It can be localized in the same way as the About page.

You may add to or style this page if you like, but **you must include all of the included attributions**. If you use any extra third-party libraries or resources, you should add your attributions to this page, otherwise, in most cases, it will be sufficient to leave this as it is.

Bundle SCAFs (offline apps only)

If your app includes offline support, you need to decide whether or not to include the SCAFs (Serverless Client Application Files) inside your app. If you do so, the app will be larger, but it will run in offline mode immediately, with no need to first update from the server.

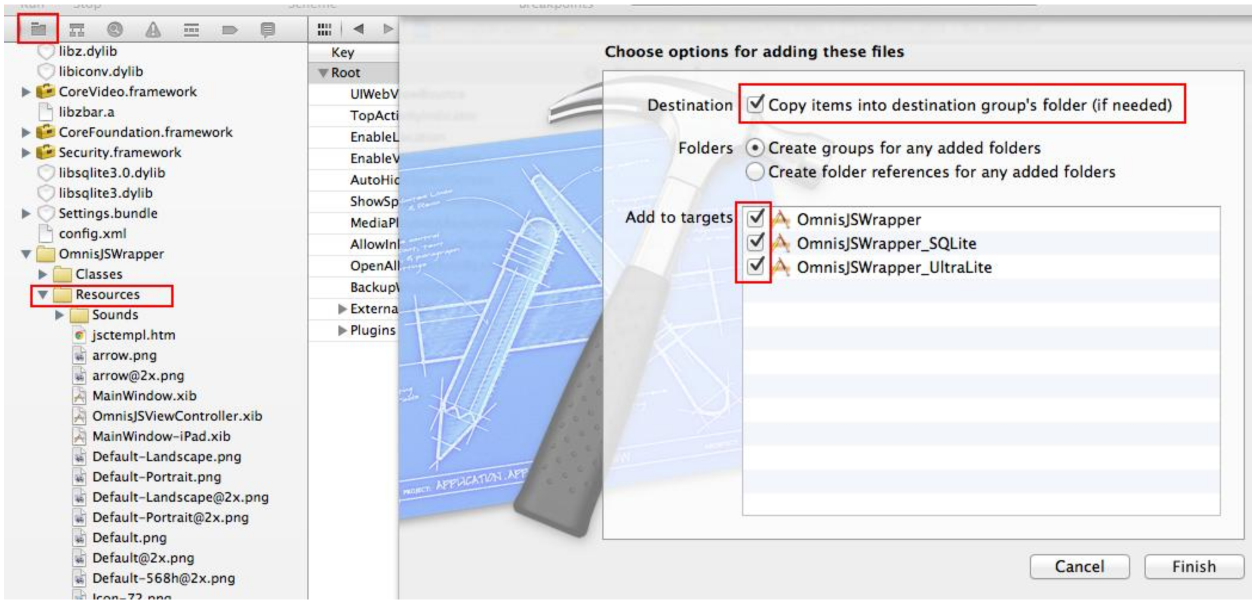
If you wish to include the SCAFs in your app, you should do the following:

- Browse to the **html/sc** folder of your Omnis Studio installation.
    - On Windows, this will be in the AppData area. e.g: *C:\Users\<username>\AppData\Local\TigerLogic\OS6.X\*

- Locate your **App SCAF** (This will be a **.db** file in the root of the *sc* folder and will be named as your library).

- Also locate your **Omnis SCAF** (This will be the **omnis.db** file in *sc/omnis/*).

- Import both of these SCAF files into your xCode project's **OmnisJSWrapper/Resources** directory.
    - The easiest way to do this is to drag them from the Finder into the project in xCode. Make sure to select the **Copy Files** option, and check the boxes to **add to all of the targets**.



Choose options for adding these files

Destination ✓ Copy items into destination group's folder (if needed)

Folders [● Create groups for any added folders] Create folder references for any added folders

Add to targets
✓ OmnisJSWrapper
✓ OmnisJSWrapper_SQLite
✓ OmnisJSWrapper_UltraLite

Bundle Local Database

It's possible to add a pre-populated SQLite database to use with your app (if you build the SQLite target). This will be used as the database which the form's *$sqlobject* connects to.

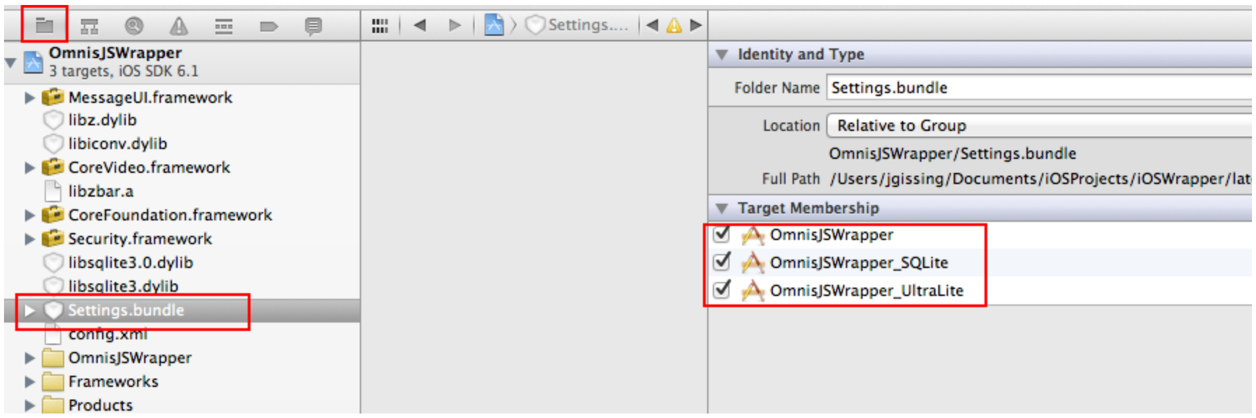
- Drag your SQLite .db file from your file system, into the **Resources** folder of your project within xCode.
    - Make sure to select the **Copy items into destination group's folder** option.
    - Also make sure that you check the box to add it to the **SQLite target**.
- Edit your project's **config.xml** file, and set the **<ServerLocalDatabaseName>** to the name of your local database (**including the .db extension**).

Bear in mind that you are creating a mobile application, and so should not be storing huge databases locally on the device.
If you need to access data from a large database, it may make sense for you to hold the whole database on your Omnis server, and use the Sync Server functionality to synchronize a subset of this data with your device.
Details on the Sync Server can be found **here**.

Remove Settings

When you are deploying your app to your end-users, you will probably not want to give them the ability to change the connection settings.
To achieve this, you need to remove the **Settings.bundle** from your project target.

- Open the *Project Navigator* view of your xCode project, and select the **Settings.bundle**.
- Un-check the target(s) you are building from the **Target Membership** for this file.

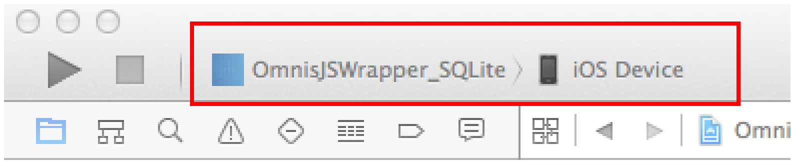


Folder Name Settings.bundle
Location Relative to Group
OmnisJSWrapper/Settings.bundle
Full Path /Users/jgissing/Documents/iOSProjects/iOSWrapper/lat

Building The App

Once you have customised the project for your application, creating a release build is very simple.

- Select the appropriate Target (based on which local database you wish to support, if any) in the *Scheme* droplist (*the left side of the highlighted area below*) at the top of the xCode window. Also select **iOS Device** as the target architecture (*the right side of the highlighted area below*).



***NOTE:** The UltraLite target requires some extra files in order to build. Please see the Building The UltraLite Target section below for details.*

- Select **Archive** from xCode's **Product** menu.
- This will create an archive of your project as it currently stands. You can then build this archive out to an Ad-Hoc or App Store App.

Building The UltraLite Target (NEW)

If you wish to synchronize your local database with another “consolidated” database, the iOS wrapper can do this using the **UltraLite** target, and using Sybase’s UltraLite & MobiLink technologies, or the same functionality can be achieved using the **SQLite** target and Omnis’ built-in Sync Server.

We recommend using the SQLite target, along with Omnis’ Sync Server, as it is a simpler process, and allows you to sync with any database backend supported by Omnis (or even others, using ODBC).

If you wish to build the UltraLite target, you will need to add the necessary UltraLite files to the project manually.

- Download and install **SQLAnywhere** to your Mac. (Version 16 has been tested)
- Browse to the installation’s **ultralite/iphone** directory and locate the **libulrt.a** file. Copy this into your project’s **dbInterface/-iphoneos** and **dbInterface/-iphonesimulator** folders.
- Browse to SQLAnywhere’s **sdk/include** directory, and copy all of the contained files into your project’s **OmnisJSWrapper/Classes/UltraLite/sdk** directory.

You should now be able to build the UltraLite target.

Building for iOS < 5.1.1 Devices

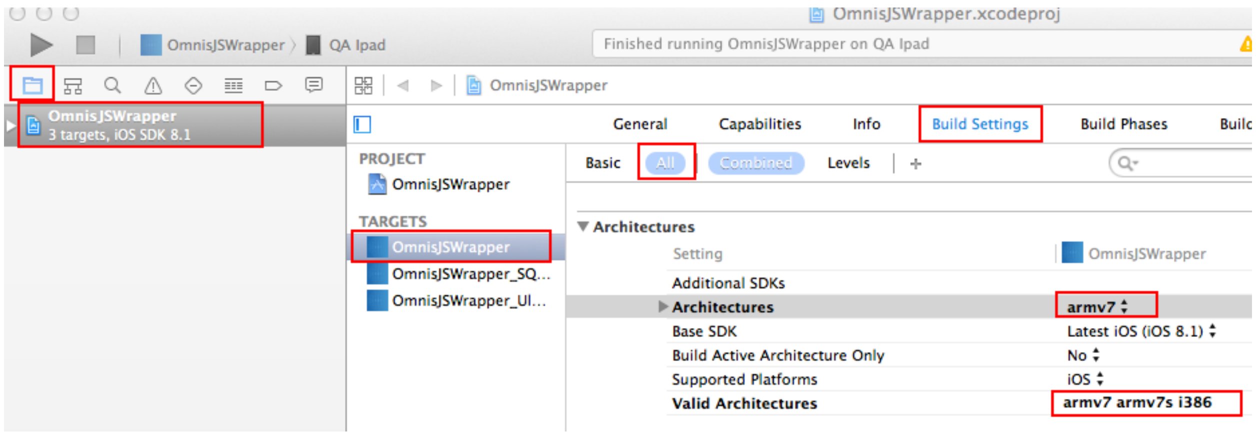
The Omnis iOS wrapper supports iOS devices as low as iOS 4.3.

However, by default the app will not run on devices running iOS versions lower than 5.1.1.
This is because Apple now stipulates that all apps submitted to the app store must include a 64-bit support. As such, this is enabled by default.

Apps compiled with a 64-bit *slice* will not run on iOS versions lower than 5.1.1.

If you wish to deploy your app to these older devices you will need to do the following (*note that this app will not be able to be submitted to the App Store*):

- Select the root of the project in the *Project Navigator*, and select the **Target** you wish to build.
- Open the target’s **Build Settings** pane, and make sure that **All** is selected at the top of the pane (as opposed to **Basic**).
- Locate the **Architectures** setting and change its value from **$(ARCHS_STANDARD)** to **armv7**.
- Locate the **Valid Architectures** setting and remove the **arm64** value.
- Build the app, as above.


Build Settings · All · Architectures

Setting	OmnisJSWrapper
Additional SDKs	
Architectures	armv7
Base SDK	Latest iOS (iOS 8.1)
Build Active Architecture Only	No
Supported Platforms	iOS
Valid Architectures	armv7 armv7s i386

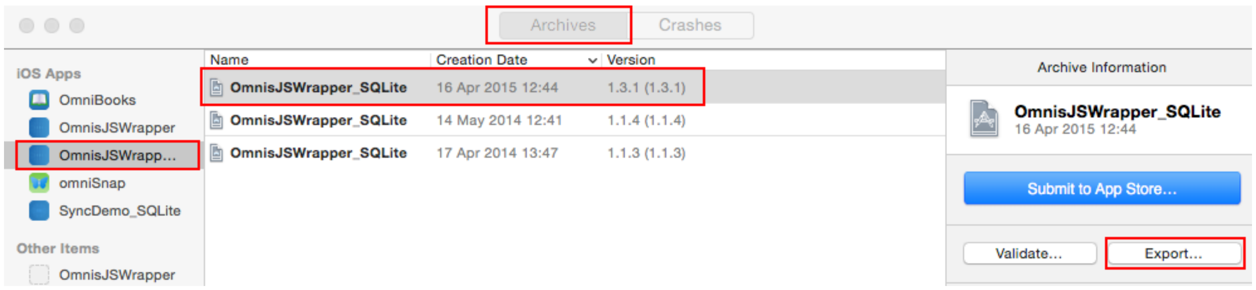
Deploying Your App

Your iOS app can be distributed manually to specific devices whose *UDID* you have added to your *Provisioning Profile*, or to any iOS device if you deploy to the App Store.

Manual Deployment

This process requires an **Ad-Hoc** provisioning profile. If you do not have one, please refer to the **Set Up Code Signing Requirements** section of this document for details on how to obtain one.

- Open xCode's **Organizer** (*from the **Window** menu*), and select the **Archives** section.
- Select the Archive which you created in the **Building the App** section of this document, then push the **Export** button.



Name	Creation Date	Version
OmnisJSWrapper_SQLite	16 Apr 2015 12:44	1.3.1 (1.3.1)
OmnisJSWrapper_SQLite	14 May 2014 12:41	1.1.4 (1.1.4)
OmnisJSWrapper_SQLite	17 Apr 2014 13:47	1.1.3 (1.1.3)

- In the window which opens, select to **Save for Ad-Hoc Deployment**, then push Next.

*If you've not already associated your Apple developer account with xCode, it will prompt you to do so. Do this using the **Add** button.*

- You will be prompted to select a **Development Team** to use for provisioning.

    - Use the droplist to select your team to automatically select a provisioning profile to sign with from the profiles stored online.
    - Or select **Use local signing assets** to automatically select a provisioning profile you have downloaded and added locally.

- Push **Export** and select the output destination in the *Export As* dialog, then push **Export**.

This will create an **.ipa** file in the location you selected in the Export dialog.

This is your signed iOS app, which is ready to be installed onto your users' devices. Distribute this file to your users, who can then install to their device by importing the .ipa file into iTunes and syncing their device.

OTA Deployment **(UPDATED)**

With a little more work than the standard Manual deployment process, you can give your users a professional, automated Over-The-Air install process via your website.

NOTE: This process requires access to a **HTTPS** web server, with a valid certificate from a provider trusted by iOS.

For testing purposes, it is possible to use Dropbox to provide secure links to your distribution files (see the notes at the end of this section).

- Begin by completing the process described in **Manual Deployment**.

- Move the generated .ipa file to a location on your **https** web server.

- Create a **.plist** file with the following content (this will be your **'manifest'**):

```
<?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>URL TO YOUR .IPA FILE</string>
                        </dict>
                        <dict>
                              <key>kind</key>
                              <string>display-image</string>
                              <key>url</key>
                              <string>URL TO A 57x57 PNG IMAGE</string>
                        </dict>
                  </array>
                  <key>metadata</key>
                  <dict>
                        <key>bundle-identifier</key>
                        <string>YOUR BUNDLE IDENTIFIER</string>
                        <key>bundle-version</key>
                        <string>APP VERSION NUMBER</string>
                        <key>kind</key>
                        <string>software</string>
                        <key>title</key>
                        <string>APP DISPLAY NAME</string>
                  </dict>
            </dict>
      </array>
</dict>
</plist>
```

- Filling in the highlighted strings with your information:
  - **software-package**: Absolute URL to your .ipa file on your web server (**must be https**).
    E.g. http://www.myserver.com/iOS_Apps/myApp.ipa
  - **display-image**: Absolute URL to a 57x57 PNG image on your server, which will display as the icon background while the app is being downloaded and installed to the device. Once installed, the app will display the icon which you defined in the XCode project.
    The Icon.png image from your project is usually suitable for this.
  - **bundle-identifier**: The **Bundle Identifier** of your application.
    E.g. com.mycompany.myapp
  - **bundle-version**: The version number of your app.
    E.g. 1.0
  - **title:** The display name you wish to use for your app while downloading. This will just be shown on the dialog asking the user if they'd like to install your app. Once installed, the app name will revert to that which you defined in xCode.

- Put this manifest file on your web server, along with your .ipa file.

- You now need to link to the manifest on a web page. You do this by create a link of the following format:
  - **itms-services://?action=download-manifest&url=<URL TO YOUR .PLIST FILE>**
  - The URL to your plist file **must be over https**.
    E.g.: <a href="itms-services://?action=download-manifest&url=https://www.mysecureserver.com/iOS/myapp.plist" >Click Me</a>

You may need to configure your web server to transmit these file types properly. This is done by setting your server's MIME Types for **.ipa** files to **application/octet-stream**, and **.plist** files to **text/xml**.

- Now, if an iOS device follows this link (and that device is included in your provisioning profile), your app will be downloaded and installed to the device wirelessly.

Don't have a suitable HTTPS server?

For testing purposes, it is possible to use Dropbox to provide secure links to your distribution files.

- Upload your .ipa file to dropbox.
- Use dropbox's web interface to generate a sharing link to the file.
    - Copy this and paste into the software-package section of your manifest (.plist file).
        - If it ends with a URL parameter (e.g. "?dl=0", remove this from the end of your URL).
    - Replace the **www.dropbox.com** part of the url with **dl.dropboxusercontent.com**
    - This edited url allows you to link directly to the file, rather than to dropbox's download interface.
- Add your manifest file to dropbox, and do the same again to get the direct link to the file to add to your html link.

App Store Deployment

DISCLAIMER: Before embarking down this route, you should read Apple's requirements and guidelines on app submission.
Tigerlogic takes no responsibility for any content of your app.

*This process requires an **App Store** provisioning profile. If you do not have one, please refer to the **Set Up Code Signing Requirements** section of this document for details on how to obtain one.*

Full details on the process can be found in **Apple's Documentation** - the information given here is an overview of the process.

The first step is to create a *record* for your app in **iTunes Connect**.

- Log in to iTunes Connect, and select **My Apps**.

- Push the **+** button at the top of the page and select **New iOS App**. Follow through the wizard to create your iOS app record. This will include setting store descriptions, images etc. The wizard provides instruction on each of these.

*Make sure your **Bundle ID** (or Bundle ID + **Suffix** if you used a wildcard Bundle ID) matches the **Bundle Identifier** of your app in xCode.*

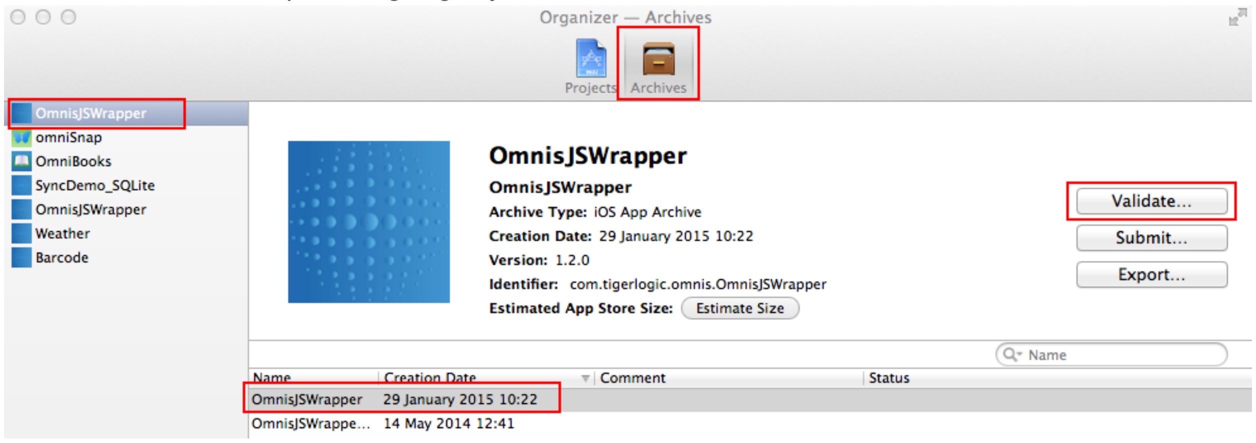
- Once you complete this, you will have an app record with a status of **Prepare For Submission**.

- You should fill in the relevant fields with details about your application, such as description, screenshots, pricing, review notes etc.

It is now possible to upload your app binary through xCode.

- Open xCode's **Organizer** (*from the **Window** menu*), and select the **Archives** section.

- Select the Archive which you created in the *Building The App* section of this document, then push the **Validate** button.
    - Follow through the wizard, selecting your team when prompted, then push **Validate**.
    - This will perform a series of checks on your app to look for any common issues which would cause your app to be rejected.
    - If this finds any issues, you will need to correct these (then create a new Archive and re-validate) before going any further.



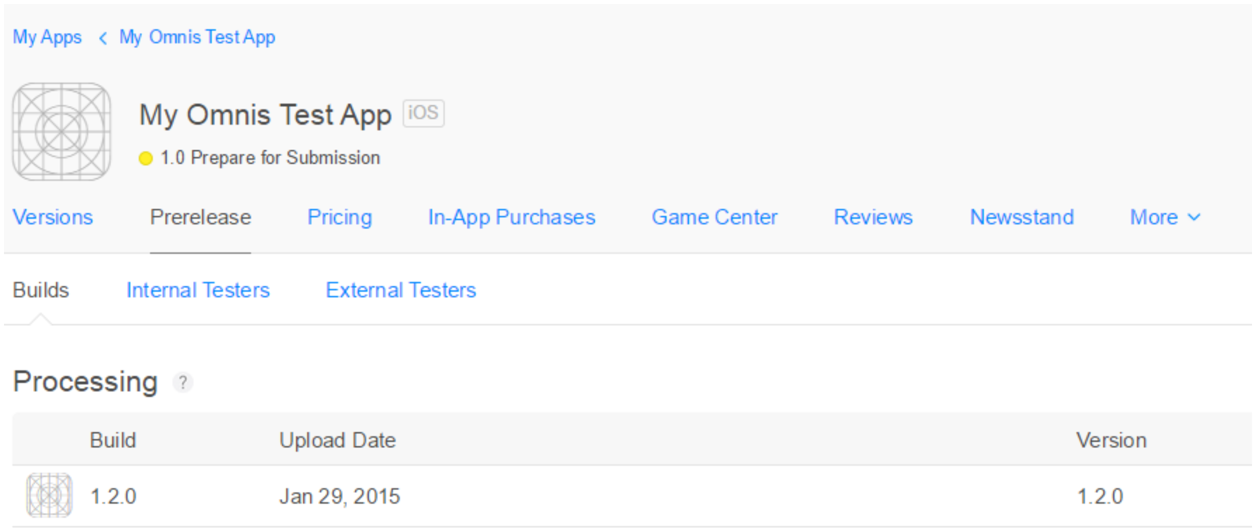
OmnisJSWrapper
OmnisJSWrapper
Archive Type: iOS App Archive
Creation Date: 29 January 2015 10:22
Version: 1.2.0
Identifier: com.tigerlogic.omnis.OmnisJSWrapper
Estimated App Store Size:

Name	Creation Date	Comment	Status
OmnisJSWrapper	29 January 2015 10:22		
OmnisJSWrappe...	14 May 2014 12:41		

- Open the Organizer and select your Archive again, but this time push the **Submit** button.

- As before, follow through the wizard, selecting your team when prompted, and finalizing by pushing the **Submit** button.

- At the end of this process, your app will have been uploaded to your iTunes Connect record, and can be seen under **Prerelease / Builds.**
    - Initially, the build will be under the heading **Processing**.
    - Once Apple have finished processing the build, it will be changed to the version number of your build, as defined in xCode.



Build	Upload Date	Version
1.2.0	Jan 29, 2015	1.2.0

- Switch back to the **Versions** pane of your app's iTunes Connect record, locate the **Build** section, and add your uploaded build.
- You can now (provided that you have filled in all of the necessary information) submit your app for review using the **Submit For Review** button at the top of the **Versions** pane.

If the review of your app is successful (this may take up to several weeks, but is usually a few days), it will become live on the App Store (unless you selected to manually release), and open to millions of potential customers.