Building The iOS Wrapper

Version 3.0.1+

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 the latest (non-beta) release of **Xcode**. You can download this (on OS X Yosemite, 10.10.4 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:

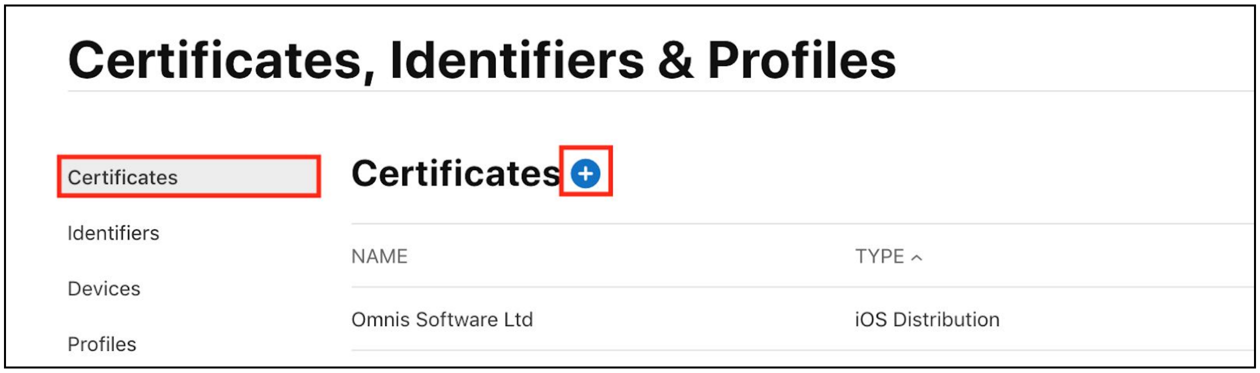
NOTE: If you wish to use Push Notifications with your app, this process differs slightly. This is explained in the **PushNotifications** document available from the wrapper download page.

However, if you are building the app for the first time, we recommend you follow the instructions below first, then add Push Notification capabilities once you are familiar with the process.

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.

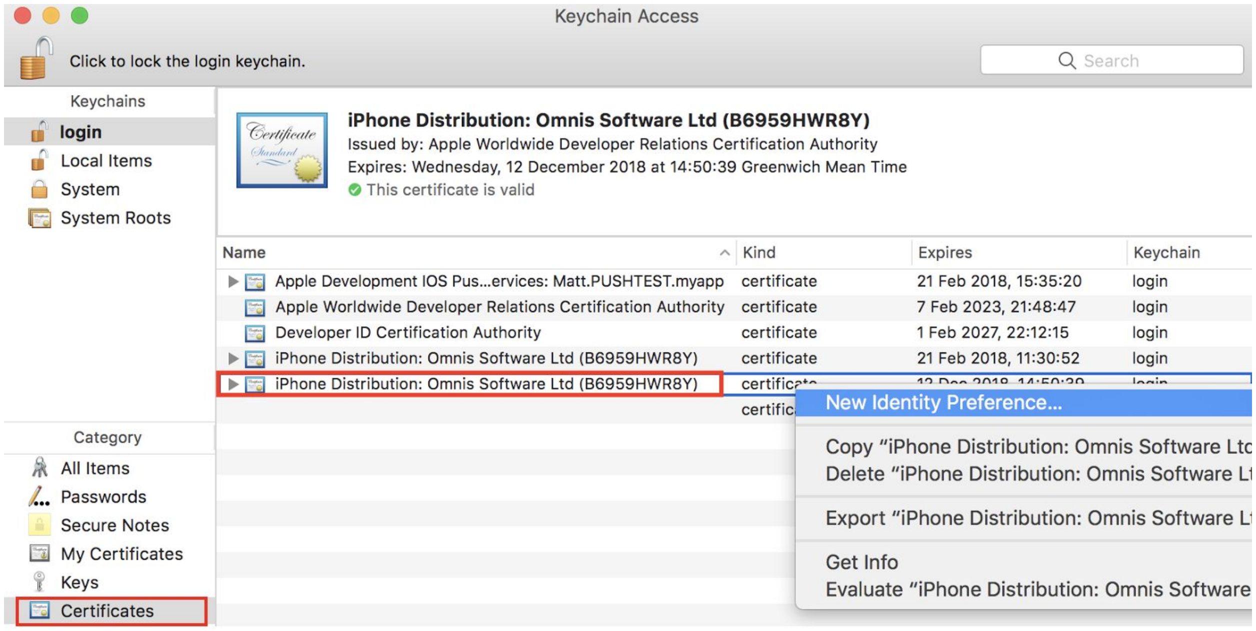


NAME	TYPE
Omnis Software Ltd	iOS Distribution

- 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.
Search
Keychains
login
Local Items
System
System Roots
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
New Identity Preference...
Copy "iPhone Distribution: Omnis Software Ltd
Delete "iPhone Distribution: Omnis Software Lt
Export "iPhone Distribution: Omnis Software Lt
Get Info
Evaluate "iPhone Distribution: Omnis Software

iPhone Distribution: Omnis Software Ltd (B6959HWR8Y)
Issued by: Apple Worldwide Developer Relations Certification Authority
Expires: Wednesday, 12 December 2018 at 14:50:39 Greenwich Mean Time
This certificate is valid

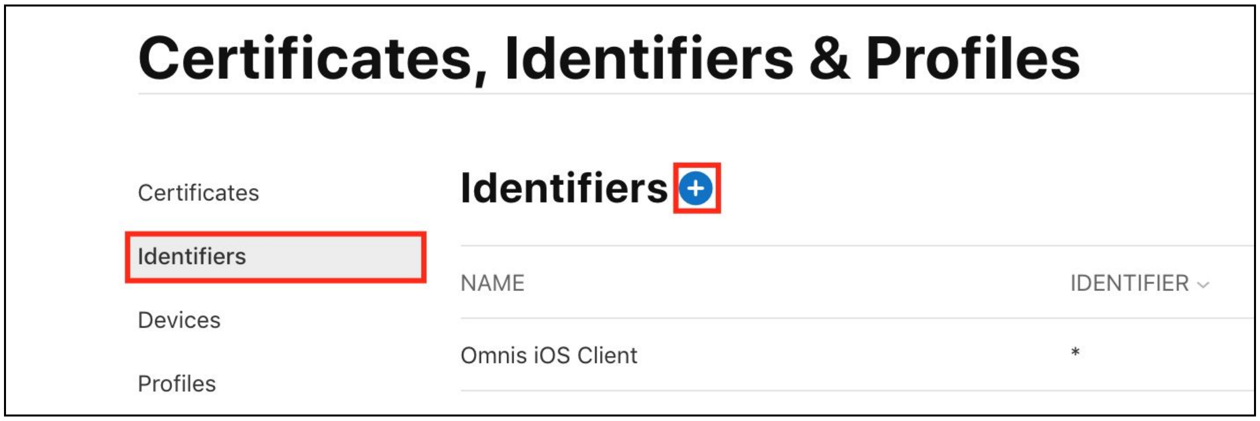
Name	Kind	Expires	Keychain
Apple Development IOS Pus...ervices: Matt.PUSHTEST.myapp	certificate	21 Feb 2018, 15:35:20	login
Apple Worldwide Developer Relations Certification Authority	certificate	7 Feb 2023, 21:48:47	login
Developer ID Certification Authority	certificate	1 Feb 2027, 22:12:15	login
iPhone Distribution: Omnis Software Ltd (B6959HWR8Y)	certificate	21 Feb 2018, 11:30:52	login
iPhone Distribution: Omnis Software Ltd (B6959HWR8Y)	certificate	12 Dec 2018, 14:50:39	login
	certific		

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.



- 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.*

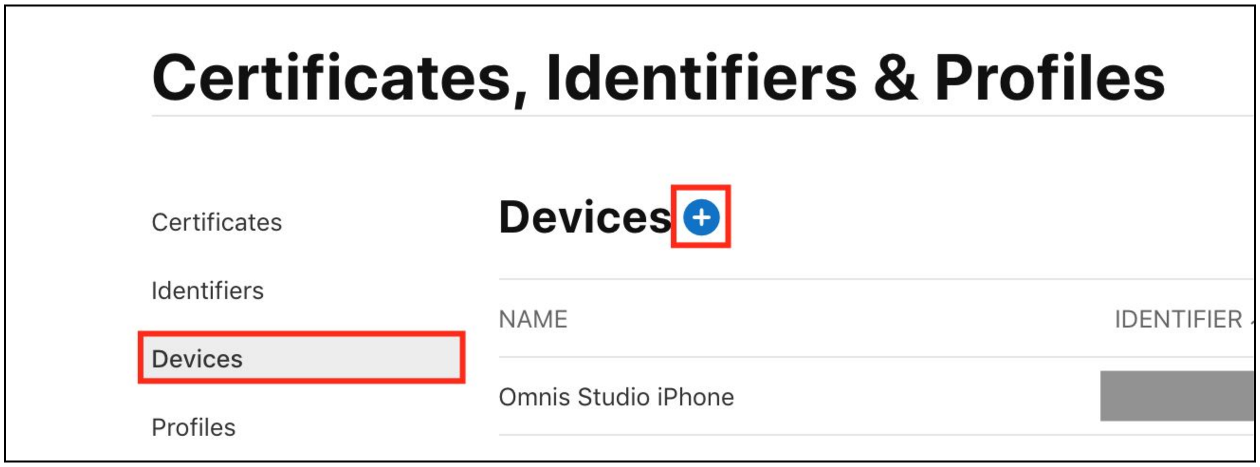
- Omnis wrapper apps do not require any *App Services*.

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.



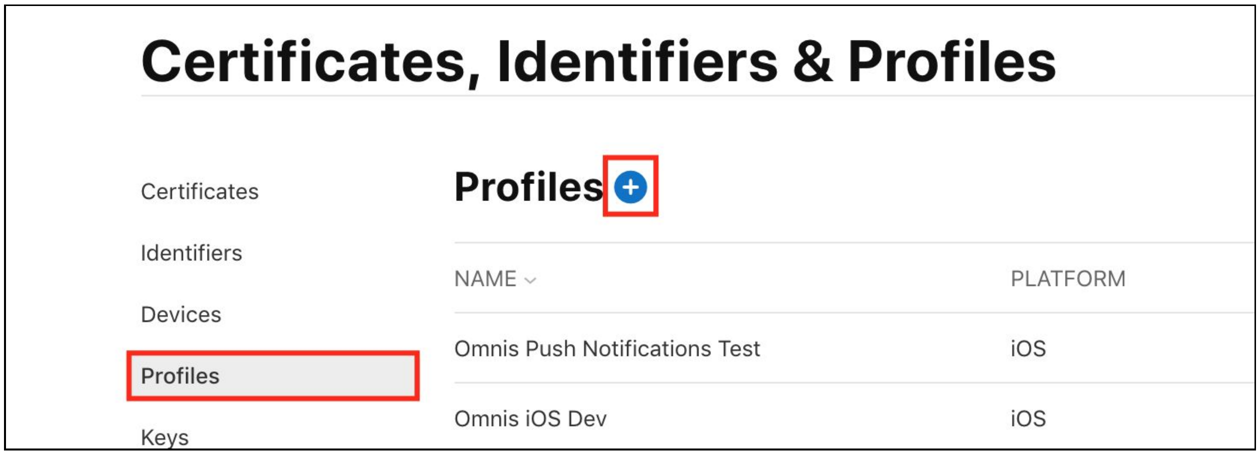
- Devices are added by their **UDID** (Unique Device Identifier). You can find your device's UDID by connecting it to iTunes, and clicking on its serial.
    - **whatsmyudid.com** provides a simple tutorial for this.

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

NAME	PLATFORM
Omnis Push Notifications Test	iOS
Omnis iOS Dev	iOS

- 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**.

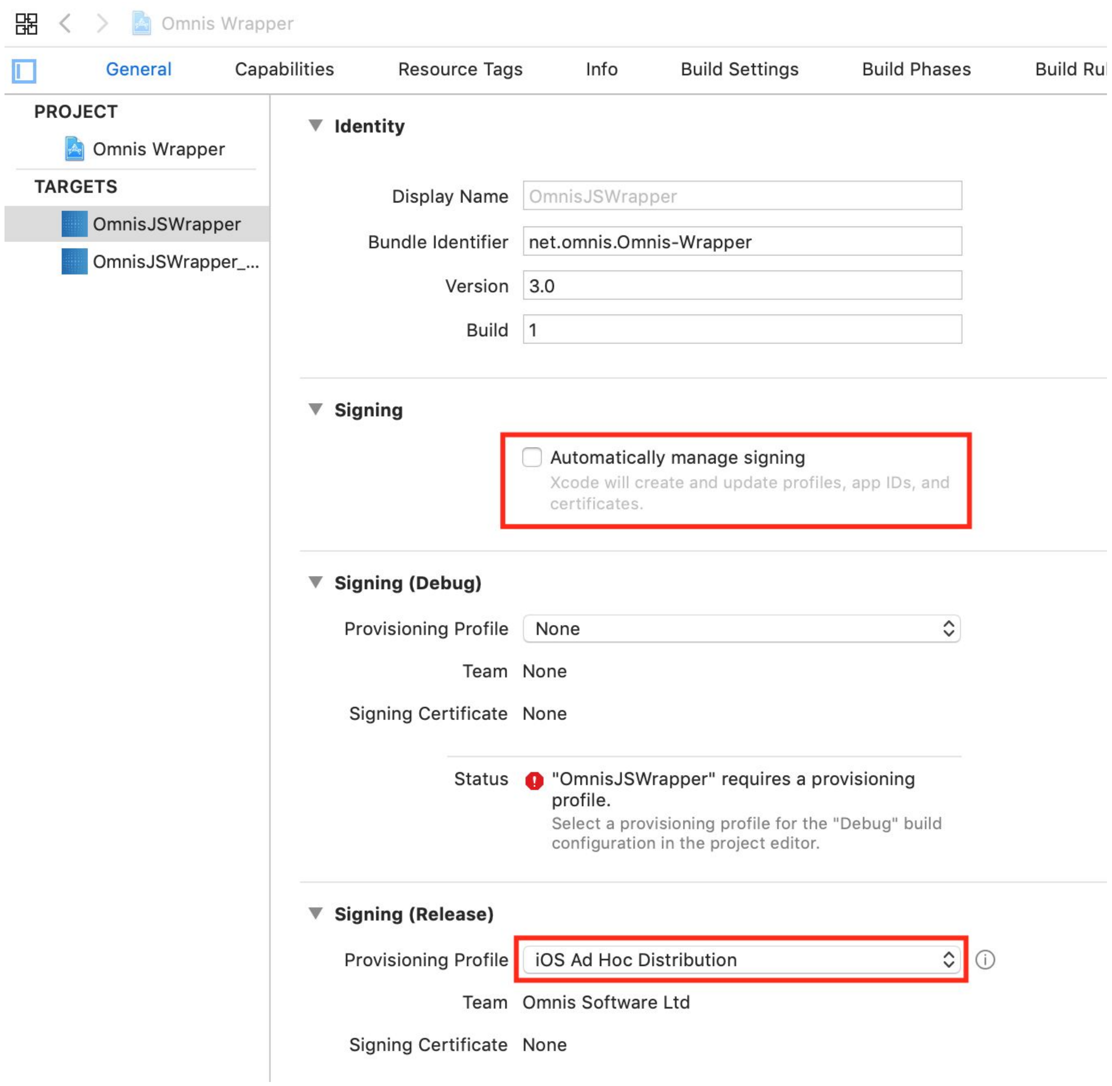
- Extract the contents of the wrapper zip file to a folder on your Mac. Make sure that there are no spaces in the path to the extracted folder.

- Double-click the **OmnisJSWrapper.xcodeproj** file, to open the project in Xcode.

Set The Project To Use Your Provisioning Profile

By default, Xcode 8+ is set to 'automatically manage signing'. For our purposes, it is easier to keep track of things if you manage this yourself.

- Select the root of your project in the *Project Navigator.*
- Under the *General* tab, make sure that "**Automatically manage signing**" is **unchecked**.
- In the **Signing (Release)** section, select your provisioning profile.
    - Note that there is no requirement for you to enter anything into the **Signing (Debug)** section, as you should not need to debug the native wrapper code.



Identity

Display Name: OmnisJSWrapper
Bundle Identifier: net.omnis.Omnis-Wrapper
Version: 3.0
Build: 1

Signing

☐ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Signing (Debug)

Provisioning Profile: None
Team: None
Signing Certificate: None
Status: "OmnisJSWrapper" requires a provisioning profile.
Select a provisioning profile for the "Debug" build configuration in the project editor.

Signing (Release)

Provisioning Profile: iOS Ad Hoc Distribution
Team: Omnis Software Ltd
Signing Certificate: None

Updating The Omnis Interface Framework (NEW)

Updates to the App Framework (containing the bulk of the features used by the wrapper) can be applied without replacing the wrapper project.

- Download an updated iOS **Omnis Interface Framework** from the wrapper download page.
- Extract the *OmnisInterface.framework* contained in the zip into the **Frameworks** folder at the root of your wrapper project.
- Clean and rebuild your app.

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:

Connection Settings

NOTE: Apple now requires all apps submitted to the App Store to support IPv6-only networking.

It seems that Apple currently still allow access to IPv4 servers on IPv6 networks which use DNS64/NAT64 translation.

So while access to IPv4 addresses should still, currently, pass App Store validation, this is clearly a move towards IPv6, and we would encourage you to make sure your server domain is registered with an IPv6 DNS.

Using hard-coded IPv4 addresses to access your server *may* cause your app to be rejected by Apple. It's advised to use domain names whenever possible.

- **WebServerURL**: 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*

- **OnlineForm**: Route to the form's .htm file from *WebServerURL*. So if you're using the built in Omnis server, it will be of the form **/jschtml/myform.htm.** If you are using a web server, it will be the remainder of the URL to get to the form, e.g. **/omnisapps/myform.htm**.

*Only **WebServerURL** & **OnlineForm** are needed for Online forms.*

Offline Mode Settings

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

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

- **AppSCAFName**: 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).

- **OmnisServer**: 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

- **OmnisPlugin**: 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

Behavior/Appearance Settings

- **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.
- **BackgroundColor**: A hexadecimal color string to specify the background color for the app. This will be visible in the status bar, and outside the 'Safe Area' on relevant devices (iPhone X etc, with a 'notch'). E.g: **#FF6699**

- **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**: Whether the pull-down menu includes the option to show the About screen. (1 for yes, 0 for no).

- **MenuIncludeCredits**: Whether the pull-down menu includes the option to show the Credits screen. (1 for yes, 0 for no).

*The **About** screen contains a link to the **Credits** screen. If MenuIncludeAbout is set to 1, you should probably set MenuIncludeCredits to 0.*

- **LocalDBName**: The name (including .db extension) of the local sqlite database to use. If you are bundling a pre-populated 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).

Push Notifications

- **PushNotificationServer**: URL to the RESTful interface to your Omnis server used for handling push notifications.
    - For direct to Omnis connections, this will be of the form: **http://<ip address>:<$serverport>**
    - When using a Web Server, this will be a URL to your RESTful web server plugin, suffixed with "**/ws/xxx**"
      Where xxx is either:
        1) nnnn (a port number)
        2) ipaddress_nnnn (IP address and port number)
        3) serverpool_ipaddress_nnnn

- **PushNotificationGroupMask**: An integer bitmask to specify the initial notification groups the app should be a member of.

Note: There is a separate document, available on the wrapper download page, detailing the use of Push Notifications. Please refer to that for more details.

NOTE: Many values in the config.xml file are marked as 'exposed' and are read only on first launch of the app. They are then saved to, and read from, local storage to allow in-app configuration from the Settings screen or your application code.
As such, if you make changes to the config.xml, affecting 'exposed' settings, you may need to uninstall the old app from your device before running the new version.

Accessing Settings From Omnis

You can use the **savepreference** & **loadpreference** *$clientcommand*s to read or set these native app settings at runtime, from your Omnis code.
To do so, you just need to prefix the setting name (the name of the XML tag in *config.xml*) with **"_SETTING_"**, and pass as the preference name when calling the above *$clientcommand*s.

E.g: `Do $cinst.$clientcommand("loadpreference",row("_SETTING_OmnisForm","iPref"))`

Creating Custom Settings

You can define your own native app settings:

- Add a new element to the **config.xml** file, at the same level as the other settings.
  - The name of this element determines the name of your setting.
- Set its **type** attribute to the appropriate data type for the setting. One of:
  - **"bool"** - The value should be **1** or **0.**
  - **"long"** - The value should be an integer.
  - **"string"** - The value should be a text string. This is the default if no **type** attribute supplied.
- Set its **exposed** attribute:
  - **"1"** - If you may want to change the value. The value will be saved to the app's user preferences. Note that newer builds of the app will not pick up changes to this in config.xml (as it is read from the user preferences after first launch).
  - **"0"** - If you want the setting to act as a **constant** that is not exposed to the user to change. Its value will be read from config.xml on each start of the app, so can be changed in newer builds. This is the default if no *exposed* attribute is set.

*Note: If a setting which is **not exposed** is ever saved using the savepreference clientcommand, it is no longer read from config.xml on startup (it will read the saved value).*

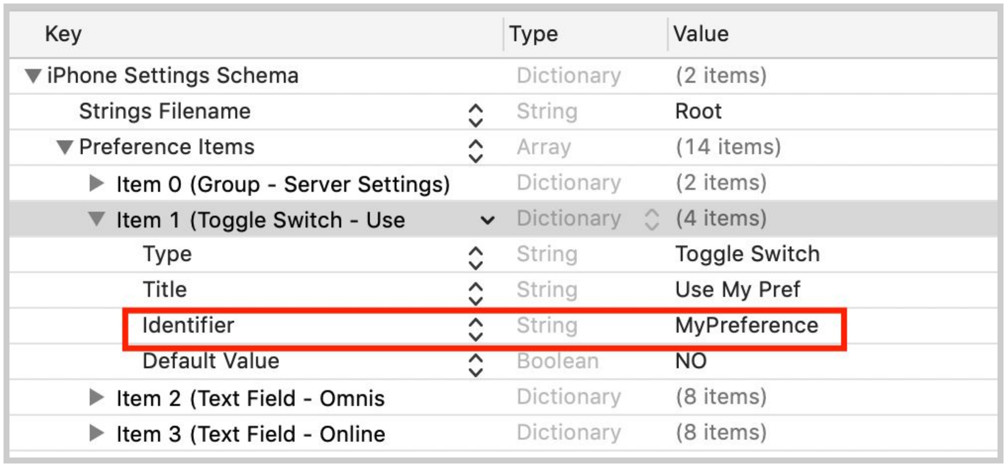
E.g:

```
<MyPreference type="bool" exposed="1">1</MyPreference>
```

Exposing/Hiding Settings For Users

If you include a **Settings.bundle** in your app, its defined entries will be shown under your app in iOS' main **Settings** app, and users can change them here.

You can add exposed settings here by editing the Settings.bundle's contained **Root.plist** file, and adding new items.
The item's **Identifier** must match the name of the related setting element in **config.xml.**



Key	Type	Value
▼ iPhone Settings Schema	Dictionary	(2 items)
Strings Filename	String	Root
▼ Preference Items	Array	(14 items)
▶ Item 0 (Group - Server Settings)	Dictionary	(2 items)
▼ Item 1 (Toggle Switch - Use	Dictionary	(4 items)
Type	String	Toggle Switch
Title	String	Use My Pref
Identifier	String	MyPreference
Default Value	Boolean	NO
▶ Item 2 (Text Field - Omnis	Dictionary	(8 items)
▶ Item 3 (Text Field - Online	Dictionary	(8 items)

Similarly, you can hide settings by removing their entry from the **Root.plist** file.
When deploying, you may wish to hide some settings (e.g. Omnis server connection settings).

Customizing The App

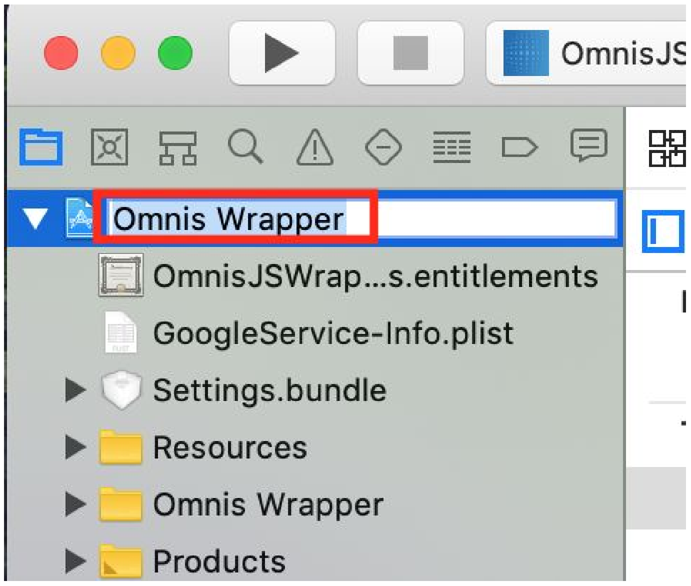
Once you have imported the wrapper project into Xcode, you should customize it for your particular application.

Rename The 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.

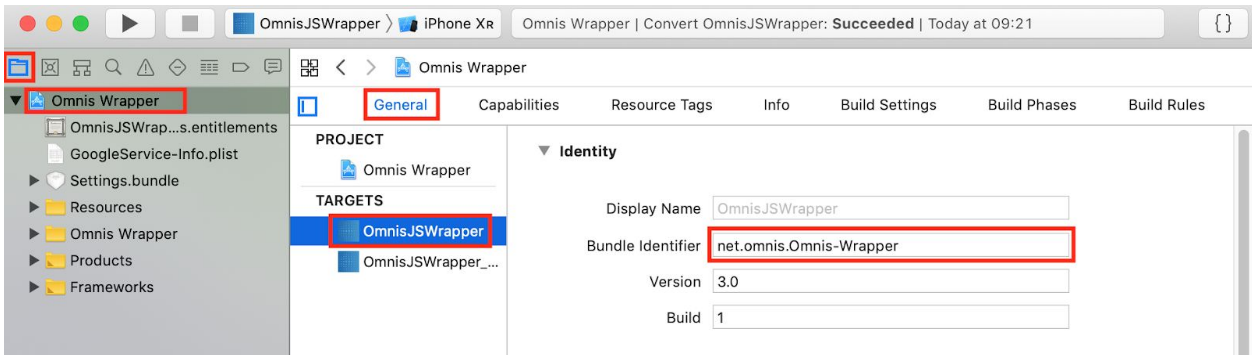


- Change the name, then push **Enter** again. You will be prompted as to whether you'd also like to rename various other instances - select to **rename all** of these.

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.


OmnisJSWrapper 〉 iPhone XR
Omnis Wrapper | Convert OmnisJSWrapper: **Succeeded** | Today at 09:21

Omnis Wrapper
- OmnisJSWrap...s.entitlements
- GoogleService-Info.plist
- Settings.bundle
- Resources
- Omnis Wrapper
- Products
- Frameworks

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

PROJECT
- Omnis Wrapper

TARGETS
- OmnisJSWrapper
- OmnisJSWrapper_...

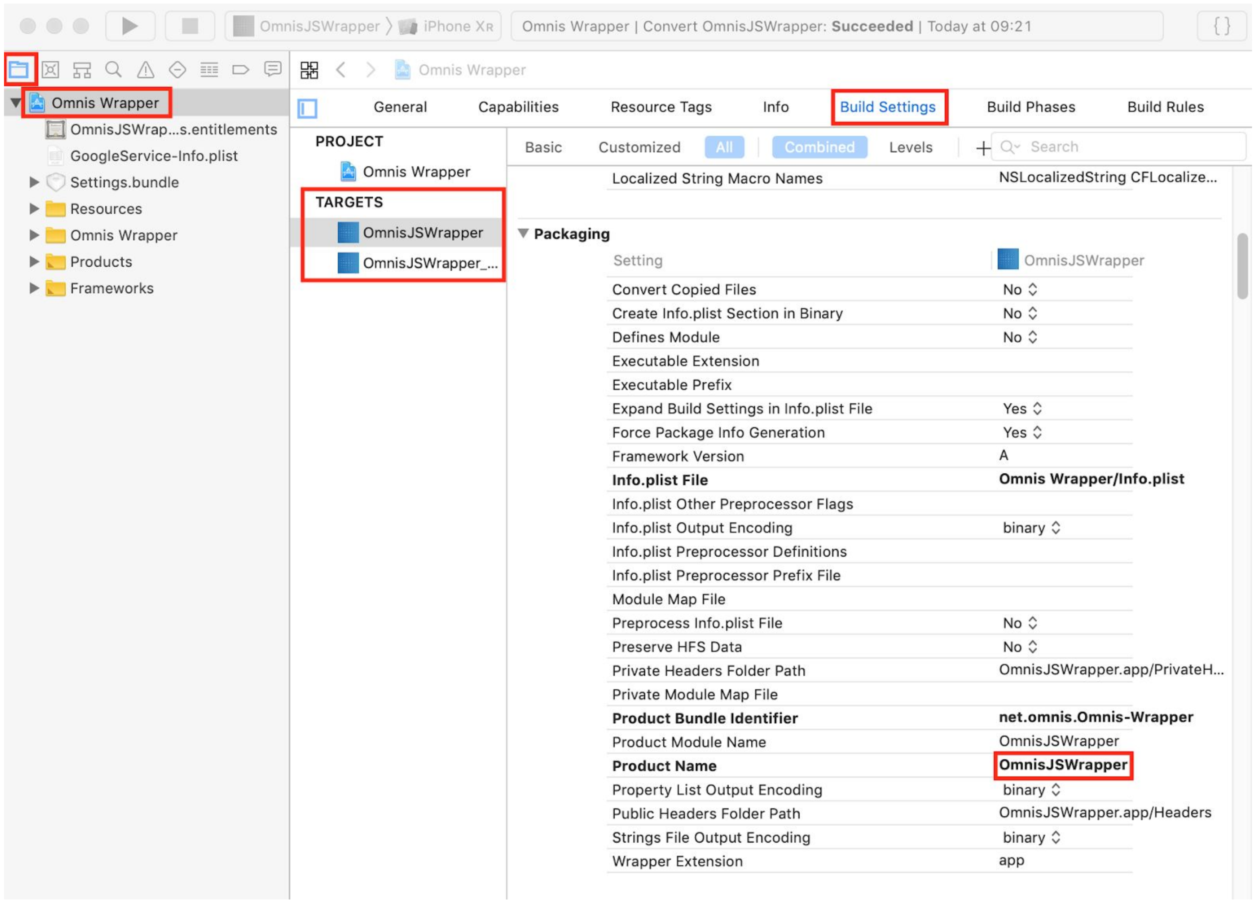
Identity

Display Name	OmnisJSWrapper
Bundle Identifier	net.omnis.Omnis-Wrapper
Version	3.0
Build	1

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.



Localized String Macro Names	NSLocalizedString CFLocalize...

Packaging

Setting	OmnisJSWrapper
Convert Copied Files	No
Create Info.plist Section in Binary	No
Defines Module	No
Executable Extension	
Executable Prefix	
Expand Build Settings in Info.plist File	Yes
Force Package Info Generation	Yes
Framework Version	A
Info.plist File	**Omnis Wrapper/Info.plist**
Info.plist Other Preprocessor Flags	
Info.plist Output Encoding	binary
Info.plist Preprocessor Definitions	
Info.plist Preprocessor Prefix File	
Module Map File	
Preprocess Info.plist File	No
Preserve HFS Data	No
Private Headers Folder Path	OmnisJSWrapper.app/PrivateH...
Private Module Map File	
Product Bundle Identifier	**net.omnis.Omnis-Wrapper**
Product Module Name	OmnisJSWrapper
Product Name	**OmnisJSWrapper**
Property List Output Encoding	binary
Public Headers Folder Path	OmnisJSWrapper.app/Headers
Strings File Output Encoding	binary
Wrapper Extension	app

Add Custom Icons & Customize Launch Screen

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

- Select **Assets.xcassets** in the *Project Navigator* (in the *Omnis Wrapper* 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.

- Select **splash_icon** in the sidebar to view the set of images used on the launch screen.

  Apple suggest that the launch screen should match the first screen of your app, rather than be used as a branded splash screen. However, we continue to show the launch screen view after startup of the app, until the webview is ready (to avoid flicker), so feel it works in this case.

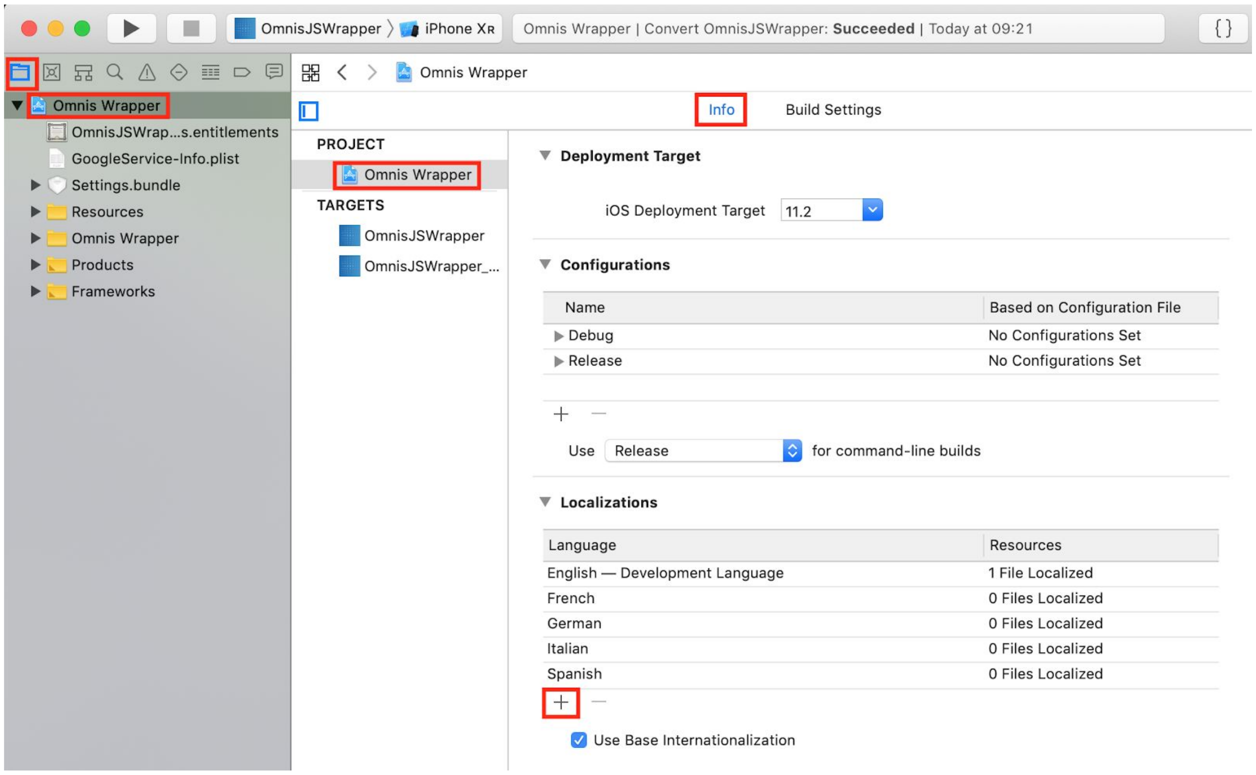
    ○ As before, you can update the images by dragging your own image files into the appropriate spaces.
    ○ You can also edit the launch screen directly, by editing **LaunchScreen.storyboard**. (E.g. for further customization such as changing background color, or if you wanted to remove the splash icon entirely).

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.



Deployment Target

iOS Deployment Target 11.2

Configurations

Name	Based on Configuration File
▶ Debug	No Configurations Set
▶ Release	No Configurations Set

Use Release for command-line builds

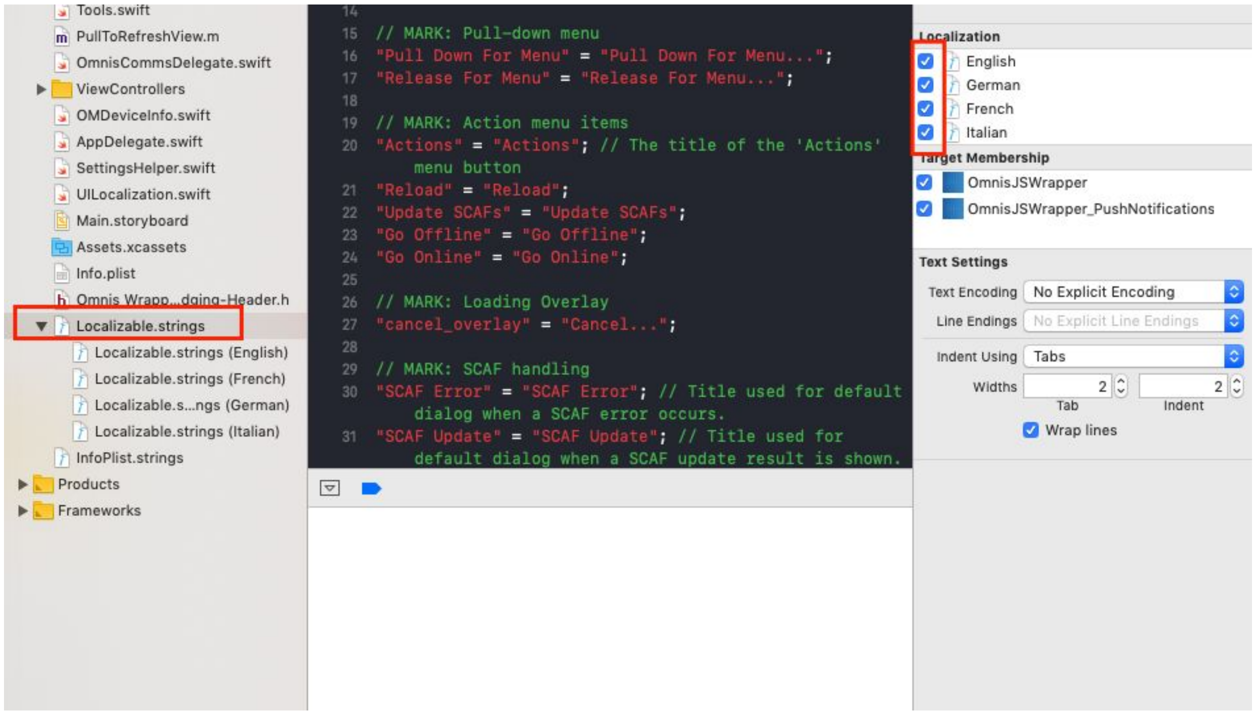
Localizations

Language	Resources
English — Development Language	1 File Localized
French	0 Files Localized
German	0 Files Localized
Italian	0 Files Localized
Spanish	0 Files Localized

✓ Use Base Internationalization

- In the window that appears, select both the **Localizable.strings** and **InfoPlist.strings** entries.

- 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.
    ○ If there is no Localizable.strings file for your language, you may need to enable it by selecting *Localizable.strings*, then checking the specific language in the **Localization** section in the right sidebar:



```
// MARK: Pull-down menu
"Pull Down For Menu" = "Pull Down For Menu...";
"Release For Menu" = "Release For Menu...";

// MARK: Action menu items
"Actions" = "Actions"; // The title of the 'Actions'
    menu button
"Reload" = "Reload";
"Update SCAFs" = "Update SCAFs";
"Go Offline" = "Go Offline";
"Go Online" = "Go Online";

// MARK: Loading Overlay
"cancel_overlay" = "Cancel...";

// MARK: SCAF handling
"SCAF Error" = "SCAF Error"; // Title used for default
    dialog when a SCAF error occurs.
"SCAF Update" = "SCAF Update"; // Title used for
    default dialog when a SCAF update result is shown.
```

Localization
- [x] English
- [x] German
- [x] French
- [x] Italian

Target Membership
- [x] OmnisJSWrapper
- [x] OmnisJSWrapper_PushNotifications

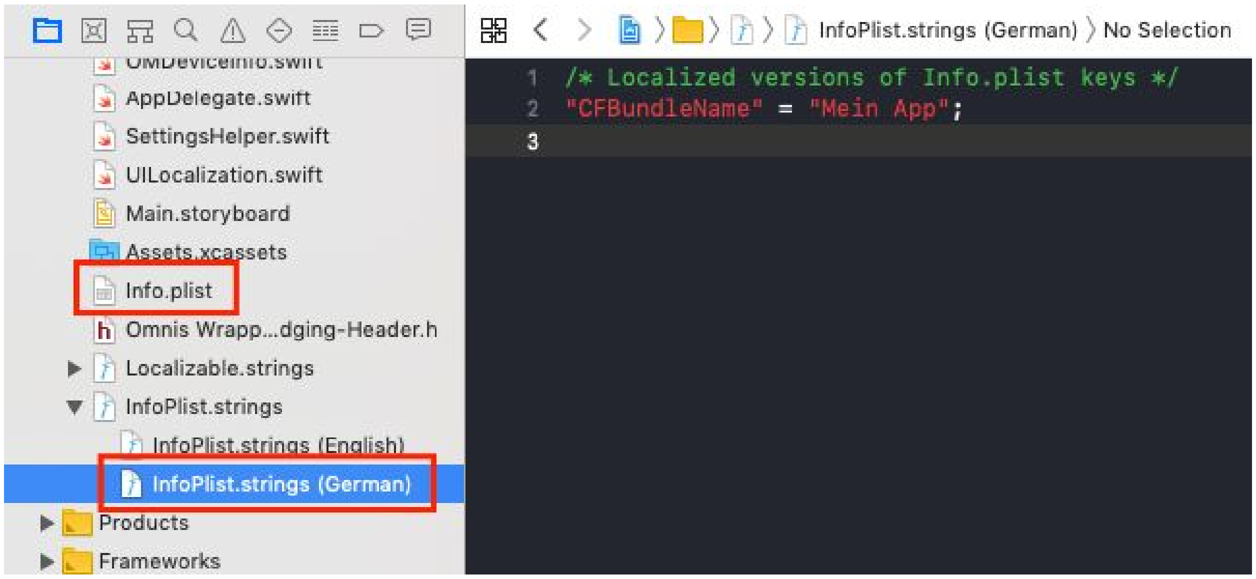
Text Settings

Text Encoding: No Explicit Encoding
Line Endings: No Explicit Line Endings
Indent Using: Tabs
Widths: 2 Tab, 2 Indent
- [x] Wrap lines

- Edit the version of the file specific to the language you are localizing for.
- 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 semicolon.

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.
    - If there's no language-specific version available, check the *Localization* checkbox in the right sidebar, as mentioned above.
- 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 **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 **semicolon**.



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

Edit The About screen

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.

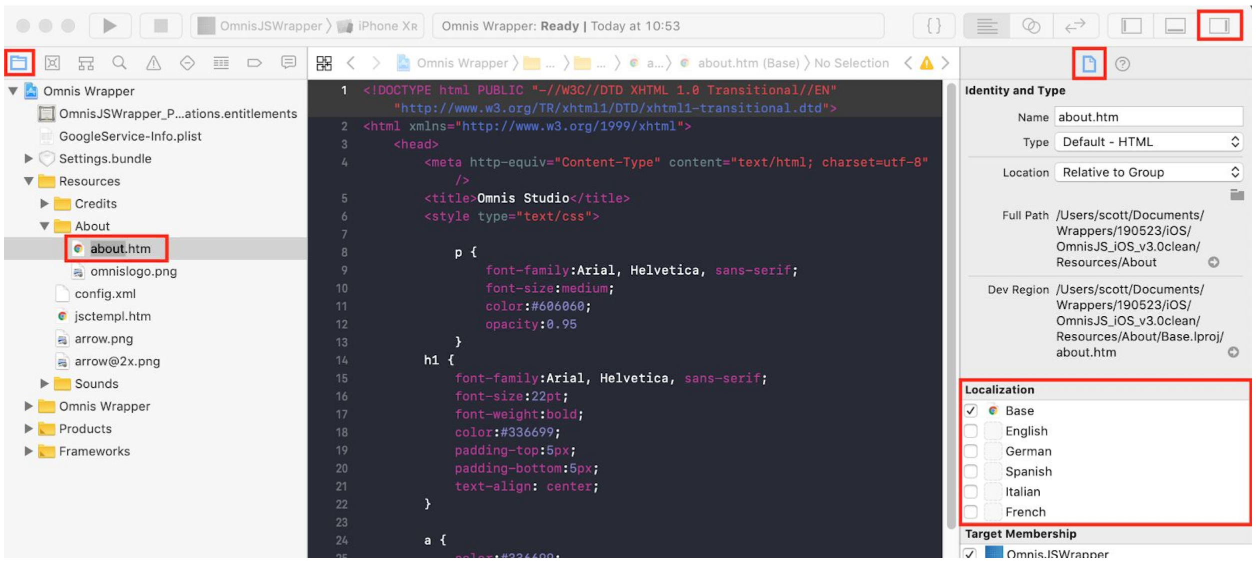
- Browse to the **Resources/About** folder in the Project Navigator.
- This folder contains **about.htm**, which is the content of the About page. You should customize this as you see fit for your application.
- Any local resources your page may need can also be added to this folder, as demonstrated 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 **Inspector** pane (the right-hand pane of Xcode), show the **file Inspector** view, and select the languages you wish to localize for in the **Localization** section.
    - If you do not see a list of languages, but instead have a "Localize…" button, you will need to click that first.

- 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:#606060;
                opacity:0.95
            }
        h1 {
            font-family:Arial, Helvetica, sans-serif;
            font-size:22pt;
            font-weight:bold;
            color:#336699;
            padding-top:5px;
            padding-bottom:5px;
            text-align: center;
        }

        a {
```

Identity and Type

Name: about.htm
Type: Default - HTML
Location: Relative to Group
Full Path: /Users/scott/Documents/Wrappers/190523/iOS/OmnisJS_iOS_v3.0clean/Resources/About
Dev Region: /Users/scott/Documents/Wrappers/190523/iOS/OmnisJS_iOS_v3.0clean/Resources/About/Base.lproj/about.htm

Localization

- ✓ Base
- ☐ English
- ☐ German
- ☐ Spanish
- ☐ Italian
- ☐ French

Target Membership

- ✓ OmnisJSWrapper

Edit The Credits Screen

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.

Edit The Offline Template HTML File

The offline forms are created from the included **jsctempl.htm** template HTML file in your project's **Resources** folder. If for any reason you want to change the HTML structure of the page, you can edit this file.

NOTE: The default jsctempl.htm file will not work with versions of Omnis prior to **10.0.1**.
If you wish to use the wrapper in offline mode with an earlier version of Omnis, replace the contents of **jsctempl.htm** with the included **jsctempl_pre10.0.1.htm** file.

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:

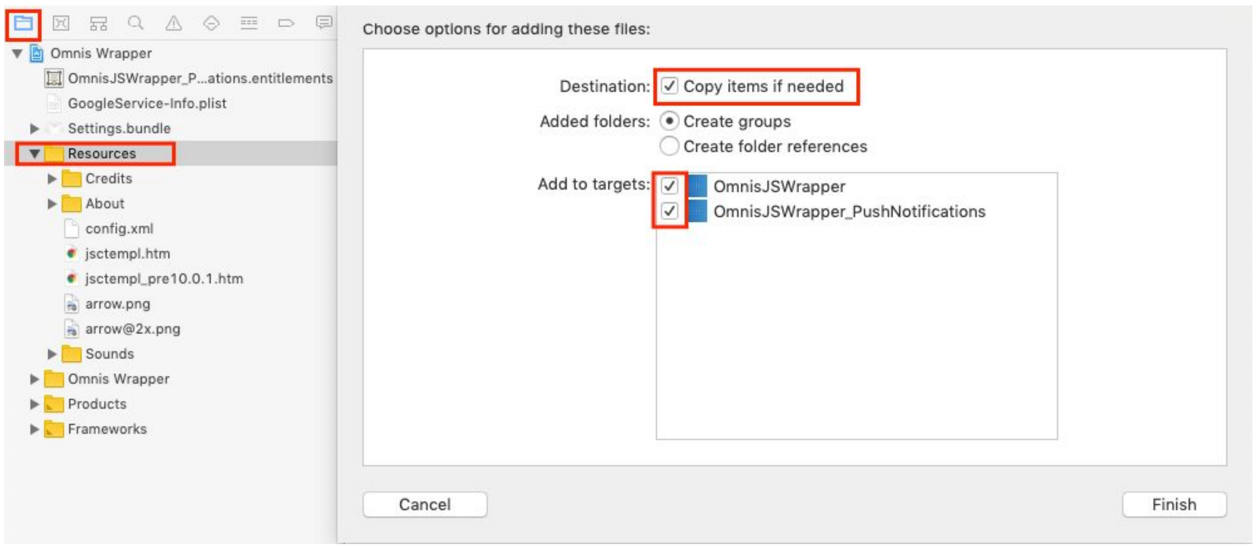
- Browse to the **html/sc** folder of your Omnis Studio installation.
  - On Windows, this will be in the AppData area. *C:\Users\<username>\AppData\Local\Omnis Software\OS10.0*
  - On macOS, this will be in the Application Support area. Macintosh HD□ ▸ □Users□ ▸ <username> ▸ □Library□ ▸ □Application Support□ ▸ □Omnis□ ▸ □OS10.0

- 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**.

- Also add your app scaf name into the config.xml as seen [here]



Choose options for adding these files:

Destination: [✓ Copy items if needed]

Added folders: [✓ Create groups] Create folder references

Add to targets: [✓ OmnisJSWrapper] [✓ OmnisJSWrapper_PushNotifications]

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 **<LocalDBName>** 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**.

Edit Settings

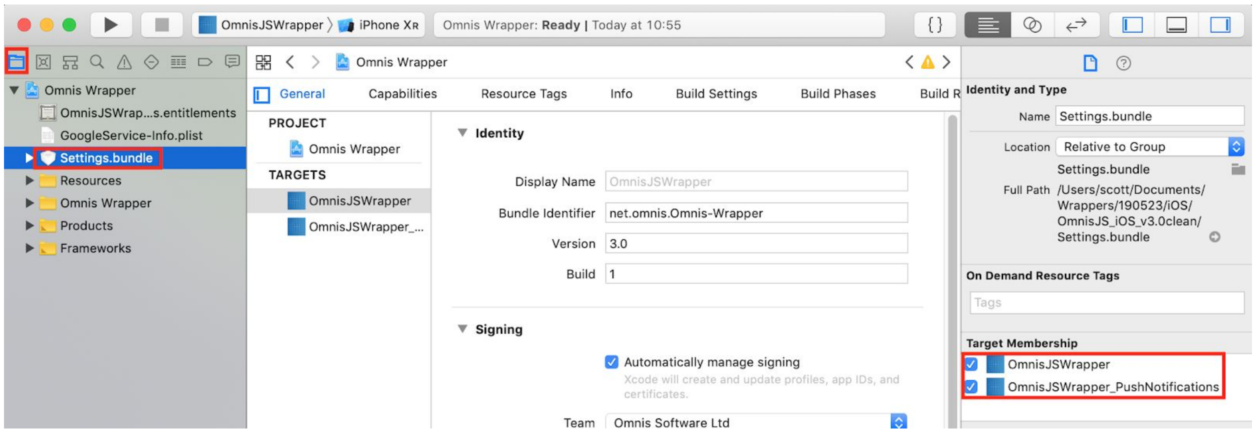
When you are deploying your app to your end-users, you will probably not want to give them the ability to change the connection settings.
To achieve this, you have two options:

1) Remove the Settings.bundle from your project target

This will completely remove the entry for your app under iOS’s **Settings** app.

- 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.



Display Name: OmnisJSWrapper
Bundle Identifier: net.omnis.Omnis-Wrapper
Version: 3.0
Build: 1

✓ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: Omnis Software Ltd

Name: Settings.bundle
Location: Relative to Group
Settings.bundle
Full Path: /Users/scott/Documents/Wrappers/190523/iOS/OmnisJS_iOS_v3.0clean/Settings.bundle

✓ OmnisJSWrapper
✓ OmnisJSWrapper_PushNotifications

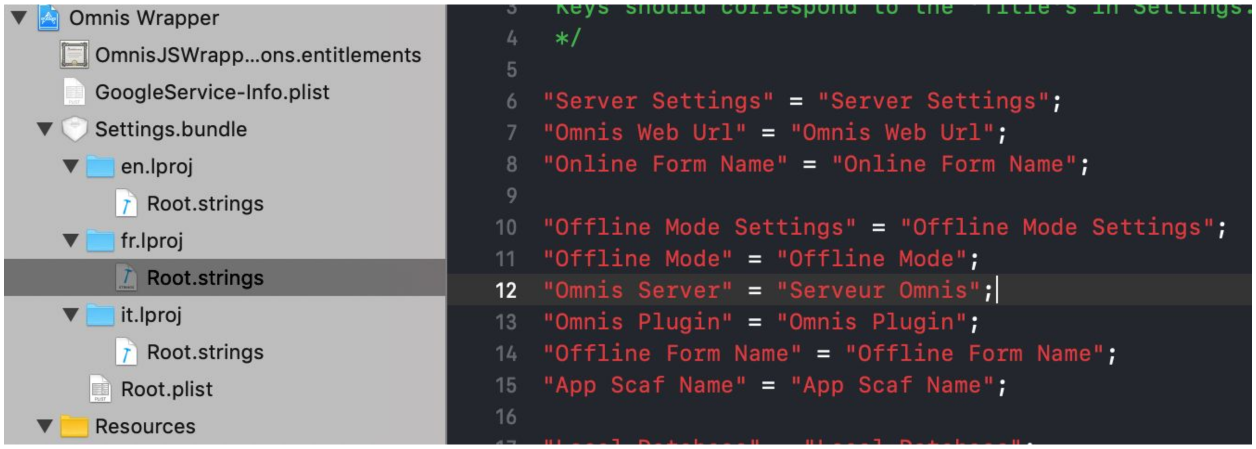
2) Remove individual settings from Settings.bundle

- Edit the **Settings.bundle**'s **Root.plist** file, and delete the *Items* related to the setting you don't wish to expose.
  - Consider taking a backup of the original Settings.bundle first.

Localize Settings

To localize the text for your settings, as shown in the Settings app (if you've not removed Settings.bundle):

- In the *Project Navigator* view, right-click on **Settings.bundle** and create a **New Folder**.
  - Name this ***<language code>*.lproj** (e.g. "fr.lproj")
- Copy the **Root.strings** file from *Settings.bundle.en.lproj* into your new folder.
- This file contains keys and translated values.
  - The **keys** (left side of the "=") correspond to the **Title** values set in *Settings.bundle/**Root.plist***
  - Add keys for any custom settings you've added, and add translations for this language as the *values* (right side of the "=") in this file.


Omnis Wrapper
OmnisJSWrapp...ons.entitlements
GoogleService-Info.plist
Settings.bundle
en.lproj
Root.strings
fr.lproj
Root.strings
it.lproj
Root.strings
Root.plist
Resources

```
*/

"Server Settings" = "Server Settings";
"Omnis Web Url" = "Omnis Web Url";
"Online Form Name" = "Online Form Name";

"Offline Mode Settings" = "Offline Mode Settings";
"Offline Mode" = "Offline Mode";
"Omnis Server" = "Serveur Omnis";
"Omnis Plugin" = "Omnis Plugin";
"Offline Form Name" = "Offline Form Name";
"App Scaf Name" = "App Scaf Name";
```

Enable App Transport Security

With iOS 9, Apple introduced **App Transport Security**, which enforces the use of secure connections from all apps.

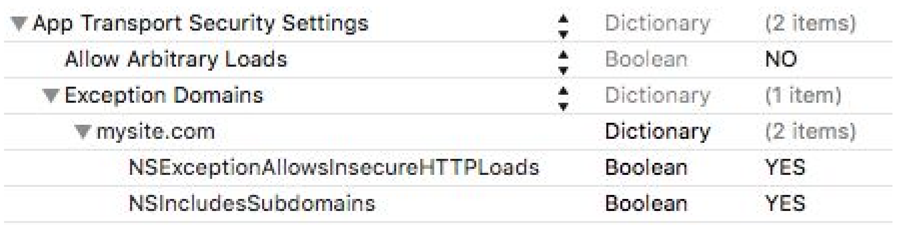
While this is a good thing for security, it does make testing your iOS app in a non-production environment tricky.
As such, the project is configured by default to allow connections of any type, to facilitate the development of your application, without requiring HTTPS web servers to be involved.

However, Apple strongly encourage (and may soon require) the use of secure connections. You should endeavor to achieve this, or at least configure the project to accept insecure connections only to a whitelist of domains, before submitting your app for App Store approval.

Control of App Transport Security is achieved through your project's **<project name>-Info.plist** file, which is located in the **Supporting Files** folder.

Documentation on the various configurations for this can be found from Apple. It is possible to achieve fine-grain control, but in short, to fall back to default behaviour and require secure connections, adhering to Apple's standards, you just need to set the value of **Allow Arbitrary Loads** to **NO**.

To enable insecure connections only for particular domains (i.e. that on which you're hosting your Omnis server/Web server), You should add a *Dictionary* of **Exception Domains**. Add a child *Dictionary* named to match your domain. Give the domain some child *Boolean* values named **NSExceptionAllowsInsecureHTTPLoads** and **NSIncludesSubdomains** (if you wish to apply to subdomains too), and set them to **YES**.



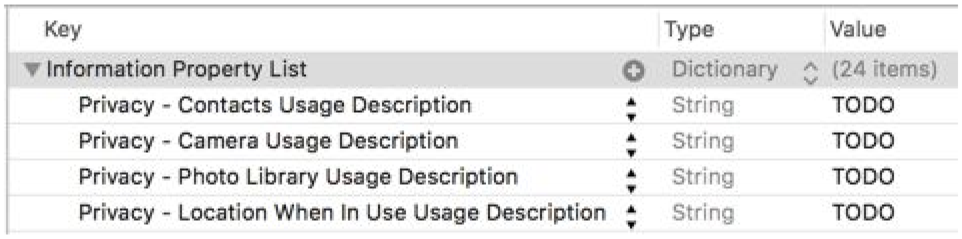
App Transport Security Settings	Dictionary	(2 items)
Allow Arbitrary Loads	Boolean	NO
Exception Domains	Dictionary	(1 item)
mysite.com	Dictionary	(2 items)
NSExceptionAllowsInsecureHTTPLoads	Boolean	YES
NSIncludesSubdomains	Boolean	YES

Set Privacy Usage Description Strings

iOS prompts the user to grant permission for the app to use certain features (location, contacts, camera etc) which could cause users concern over their privacy and how the app is using the information.

The app's Info.plist allows (and iOS 10+ *requires*) you to provide text describing how your app uses each of these features. This text will be presented to the user with the prompt to grant the app permission to access that feature.
By default, these text strings are set to "TODO", so it's important that you populate these with useful strings which describe how your app uses this data.

- Open your project's **<project name>-Info.plist** file, which is located in the **Supporting Files** folder.
- At the top of the file you will find several "**Privacy - <XXX> Usage Description"** keys. Set the values of these to a description of how your app uses that particular feature.

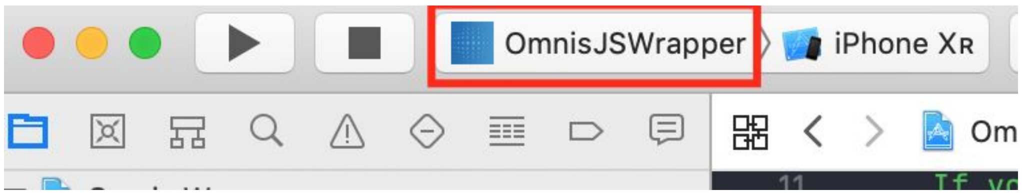


Key	Type	Value
Information Property List	Dictionary	(24 items)
Privacy - Contacts Usage Description	String	TODO
Privacy - Camera Usage Description	String	TODO
Privacy - Photo Library Usage Description	String	TODO
Privacy - Location When In Use Usage Description	String	TODO

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 whether you wish to use Push Notifications) 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 PushNotifications target requires some extra setup in order to build. Please see the Push Notifications document on the wrapper download page for more 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.

Deploying Your App

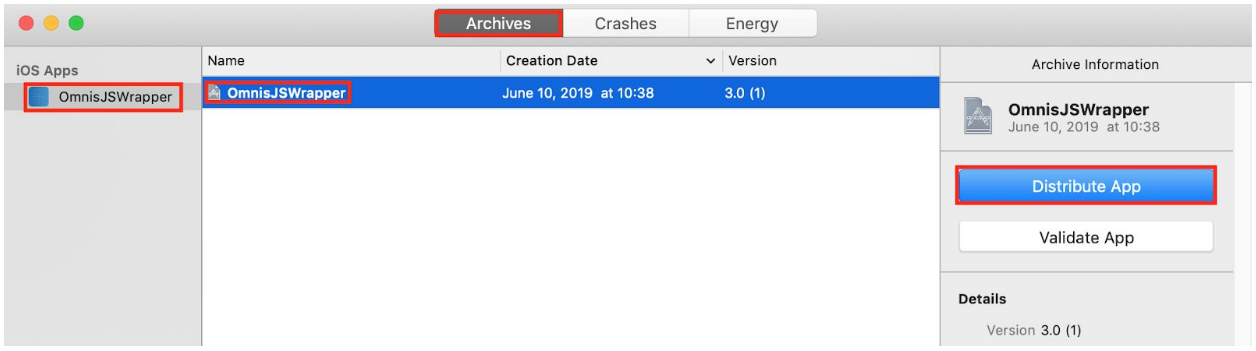
Your iOS app can be distributed manually to specific devices whose *UDID* you have added to your *Provisioning Profile*, or to any iOS device if you deploy to the App Store.

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 **Distribute App** button.



- In the window which opens, select to **Ad-Hoc**, 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.

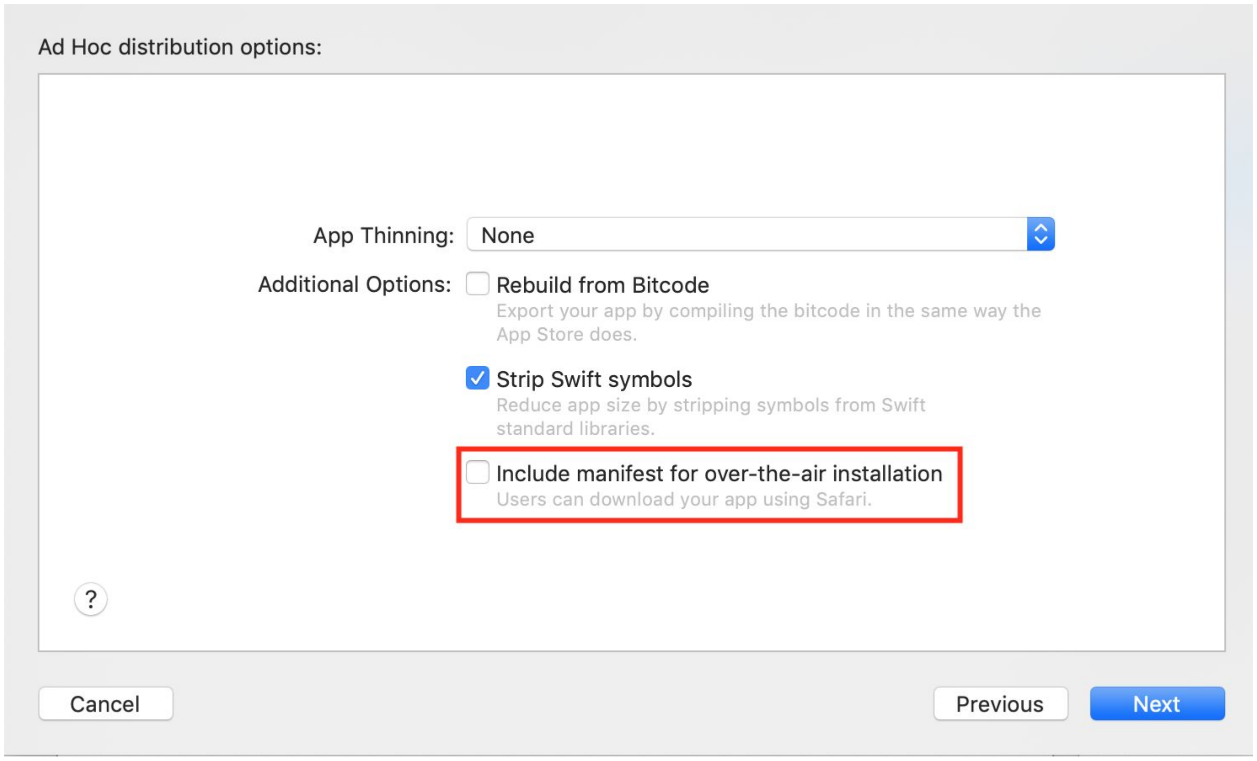
OTA Deployment

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.
Apple's documentation on this can be found at: http://help.apple.com/deployment/ios/#/apda0e3426d7 if you would prefer a more detailed description than that described here.

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 following the process described in **Manual Deployment**, when you reach the following screen, check the **Include manifest for over-the-air installation**.


Ad Hoc distribution options:

App Thinning: None

Additional Options:

- [ ] Rebuild from Bitcode
  Export your app by compiling the bitcode in the same way the App Store does.
- [x] Strip Swift symbols
  Reduce app size by stripping symbols from Swift standard libraries.
- [ ] Include manifest for over-the-air installation
  Users can download your app using Safari.

- On the next screen you'll be prompted for the following values:

    - **Name:** 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.
    - **App URL**: 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 URL**: 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.
    - **Full Size Image URL**: Absolute URL to a 512x512 PNG image on your server, which will be used to display the app in iTunes.

- Move the generated .ipa and manifest files to a location on your **https** web server.

- You now need to link to the manifest on a web page. You do this by creating 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.
Omnis Software Ltd 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.*

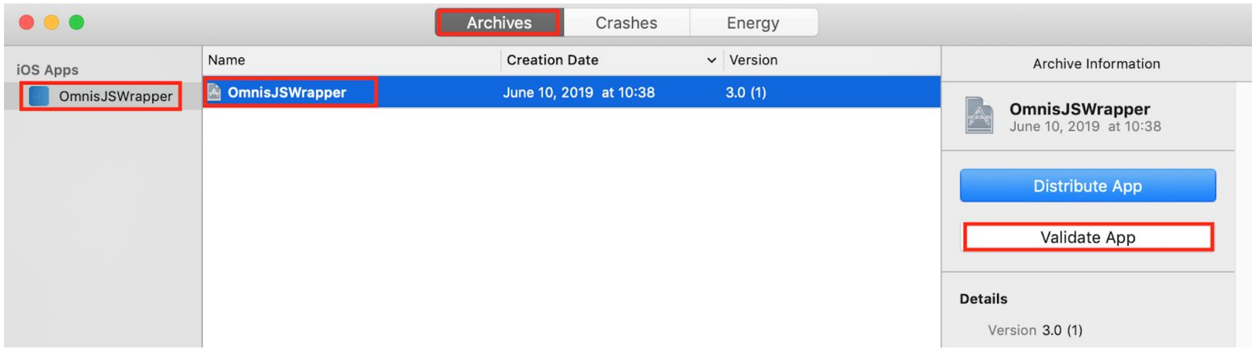
- Once you complete this, you will have an app record with a status of **Prepare For Submission**.

- 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.



Name	Creation Date	Version
OmnisJSWrapper	June 10, 2019 at 10:38	3.0 (1)

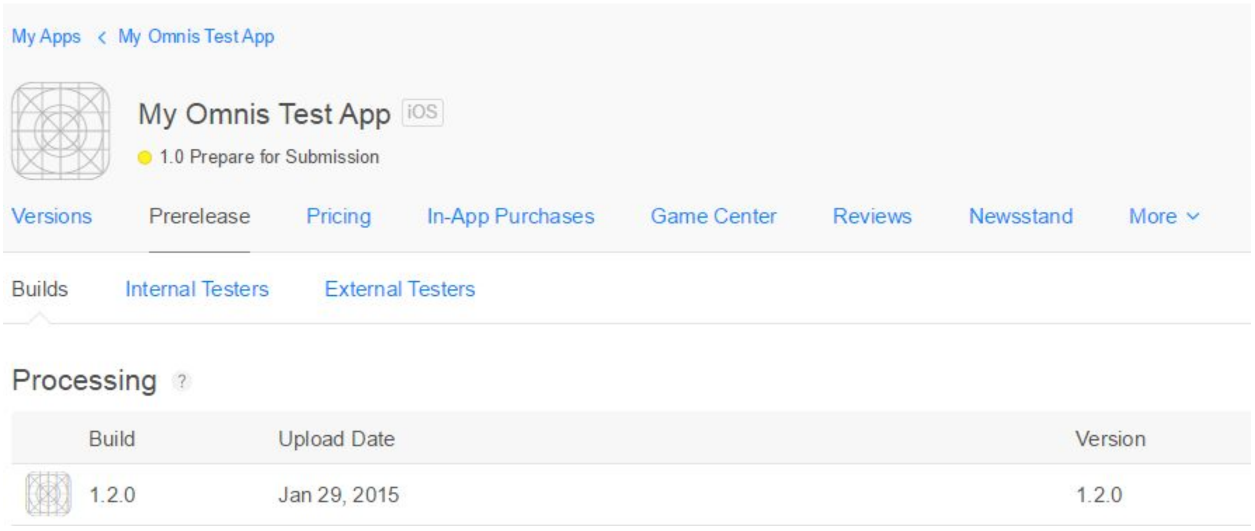
OmnisJSWrapper
June 10, 2019 at 10:38

Version 3.0 (1)

- 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.



Processing ?

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.