PayPal™

Mobile Express Checkout Library Developer Guide and Reference – iOS Edition

Last updated: January 2011

PayPal Mobile Payments Developer Guide and Reference – iOS Edition

Document Number 10123.en_US-201011

Contents

Preface

The PayPal Mobile Express Checkout Library lets you embed your mobile implementation of Express Checkout in mobile applications for the iPhone® and iPod touch®.

Purpose

The PayPal Mobile Express Checkout Library lets you embed Express Checkout in iPhone and iPod touch applications. Download the library from x.com/mobile and include it in your application. You need only a few lines of code to integrate the library and your Express Checkout implementation in your mobile application.

Scope

This document describes how to use the PayPal Mobile Express Checkout Library to embed your mobile implementation of Express Checkout in your mobile application. It also describes how to provide your build to PayPal. We review your application so we can approve it to accept payments through the library.

Revision History

The following table lists revisions made to the *Mobile Express Checkout Library Developer Guide and Reference*.

Date Published	Description
January 2011	Updated to reflect the disabling of 'Keep Me Logged In' functionality for version 1.0.1
November 2010	Created for version 1.0 of the Mobile Express Checkout Library

Where to Go for More Information

- Express Checkout Integration Guide
- Mobile Payments Library Developer Guide and Reference – iOS Edition
- Sandbox User Guide
- Merchant Setup and Administration Guide
- PayPal X Developer Network (x.com)

1. PayPal Mobile Express Checkout Library

This section provides details about the Mobile Express Checkout Library. It provides instructions and examples for using the library to embed your mobile implementation of Express Checkout in your iPhone or iPod touch application.

Mobile Express Checkout Library API Reference

The library supports 2 programming flows. They differ in where you place the PayPal button that buyers click to begin checking out with PayPal.

Programming Flow with the PayPal Button in Your Mobile Application

Place the PayPal button in your mobile application if your checkout process begins and ends with screens in your mobile application. In this programming flow, you embed only the mobile Express Checkout payment pages in a web view.

1. Fetch a device token from the library, just before you display the mobile application screen where you show a PayPal button.

   Include a pointer to your delegate method that receives the device token.

2. Get a PayPal button from the library, and place it on your mobile application screen.

   Include a pointer to your delegate method that handles the button-click event.

3. When buyers select the PayPal button, your delegate method is called:
   a. Call a routine on your mobile web server, passing the payment information.
   b. On your mobile web server, send a `SetExpressCheckout` request with the payment information to PayPal.
   c. Pass the checkout token returned in the `SetExpressCheckout` response from your mobile web server to your mobile application.
   d. Open a web view, and redirect the browser to PayPal with the mobile command, the device token, and the checkout token as URL parameters.

   ```
   https://www.paypal.com/cgi-bin/webscr?cmd=_express-checkout-
   mobile
   &drt=valueFromMobileExpressCheckoutLibrary&token=valueFromSetExpr
   essCheckoutResponse
   ```

4. Monitor the web view for a redirect from PayPal to your return or cancel URL.

5. If PayPal redirects the web view to your return URL, call surrogate routines on your mobile web server that send `GetExpressCheckoutDetails` and `DoExpressCheckoutPayment` requests to PayPal to complete the payment.

IMPORTANT: Never send Express Checkout requests from your mobile application directly to PayPal. The requests require your PayPal API credentials. Placing your credentials on mobile devices exposes you and PayPal to unacceptable security risks. Send Express Checkout requests only from secure servers.

Programming Flow with the PayPal Button on Your Mobile Website

Place the PayPal button on your mobile website if your checkout process begins and ends with pages on your mobile website. In this programming flow, you embed your entire mobile Express Checkout implementation in a web view.

1. Fetch a device token from the library, just before you open a web view of your mobile Express Checkout implementation.

   Include a pointer to your delegate method that receives device tokens.

2. Open a web view of a page or routine on your mobile web server that begins your checkout process.

   Include the device token as a URL parameter when you open the web view, along with the item details in the shopping cart.

3. Monitor the web view for a redirect from your web server to a well-known URL that signals the checkout process on your mobile website is complete.

Methods in the Mobile Express Checkout Library

fetchDeviceReferenceTokenWithAppID Method

The `fetchDeviceReferenceTokenWithAppID` method returns a device token. Use the `del` parameter to specify your own delegate function of that receives device tokens. Include the device token as the `&drt` parameter in the URL when your redirect the buyer's mobile browser to PayPal. Device tokens expire after 45 minutes.

By default, the library uses PayPal's production servers for fetching device tokens. To test your application, use the optional `env` parameter so the library fetches device tokens from PayPal's Sandbox servers, instead.

In your programming flow, fetch the device token just before you get the PayPal button.

```
- (void)fetchDeviceReferenceTokenWithAppID:(NSString const *)inAppID
forEnvironment:(PayPalEnvironment)env
withDelegate:(id<DeviceReferenceTokenDelegate>)del;
```

Parameter	Description
`inAppId:`	*(Required)* PayPal Application ID from X.com. For the Sandbox environment, use APP-80W284485P519543T.
`env:`	*(Optional)* Which PayPal servers the library uses Allowable values are: • `ENV_LIVE` • `ENV_SANDBOX` • `ENV_NONE` **NOTE:** The `ENV_LIVE` environment does not support simulators.
`del:`	*(Required)* Your delegate function that receives device tokens

getPayButtonWithTarget Method

If you place the PayPal button in your mobile application, get an instance from the Mobile Express Checkout Library. This method returns a `UIButton` that you place on your mobile application screen. If you need to move the button, because your application supports rotation for example, change the button frame. The target parameter sets which `UIViewController` receives delegate callbacks. If data is invalid, you receive an alert from the `UIAlertViews`.

```
- (UIButton *)getPayButtonWithTarget:(NSObject const *)target
andAction:(SEL)action andButtonType:(PayPalButtonType)inButtonType;
```

Parameter	Description
`target:`	*(Required)* The `UIViewController` that is the delegate for callbacks
`action:`	*(Required)* Your method that responds to the PayPal button click
`buttonType:`	*(Required)* Size and appearance of the PayPal button Allowable values are: • `BUTTON_152x33` • `BUTTON_194x37` • `BUTTON_278x43` • `BUTTON_294x43`

getInstance Method

The library provides a singleton instance of the PayPal object. Use the `getInstance` method to set and access runtime properties of the library. For example, use the value of the `paymentsEnabled` property to determine whether your attempt to fetch a device token was successful.

```
+ (PayPal*)getInstance;
```

The following table lists the properties of the PayPal object that you are most likely to use in your mobile application. For a complete list of properties, see the `PayPal.h` file.

Property	Description
`lang`	Locale code for the label of the PayPal button. By default, the library uses the locale of the device.
`errorMessage`	If the library fails to acquire a valid device token, the error message provides more details about the failure.
`paymentsEnabled`	If your attempt to fetch a device token succeeded, the value of this property is `TRUE`.

Enumerated Values in the Mobile Express Checkout Library

The enumerated values supported by methods in the library are:

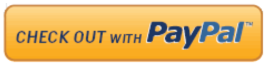
PAYPAL_ENVIRONMENT

- **`ENV_LIVE:`** Use the PayPal production servers to obtain device tokens. This environment does not support simulators.
- **`ENV_SANDBOX:`** Use the PayPal testing servers to obtain device tokens.
- **`ENV_NONE:`** Do not use any PayPal servers to obtain device tokens.

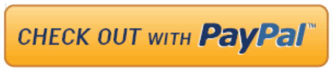
PAYPAL_BUTTON_TYPE

PayPal displays the following images for buyers on the mobile device.

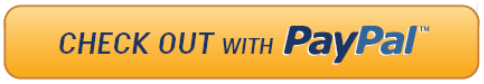
- **`BUTTON_152x33:`**



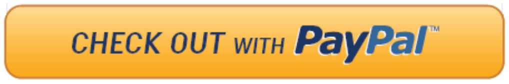
- **`BUTTON_194x37:`**



- **`BUTTON_278x43:`**



- **`BUTTON_294x43:`**



Localization Support in the Mobile Express Checkout Library

The Mobile Express Checkout Library supports many locales. Set the locale after you initialize the library. The default is the locale of the device. If the library does not support the device locale, the library uses `en_US`, instead.

How to Set the Language and the Region

Set the locale with the `lang` property of the PayPal object. Set this property any time after you initialize the library, which occurs when you fetch a device token. Set the `lang` property before you call the `getPayButtonWithTarget()` method to obtain a localized PayPal button.

Locales Supported by the Mobile Express Checkout Library

The `lang` property of the PayPal object allows these values.

Country or Region	Supported Locale Codes
Argentina	`es_AR`
Brazil	`pt_BR`
Australia	`en_AU`
Belgium	`en_BE nl_BE fr_BE`
Canada	`en_CA fr_CA`
France	`fr_FR en_FR`
Germany	`de_DE en_DE`
Hong Kong	`zh_HK en_HK`
India	`en_IN`
Italy	`it_IT`
Japan	`ja_JP en_JP`
Mexico	`es_MX en_MX`
Netherlands	`nl_NL en_NL`
Poland	`pl_PL en_PL`
Singapore	`en_SG`
Spain	`es_ES en_ES`
Switzerland	`de_CH en_CH fr_CH`
Taiwan	`zh_TW en_TW`
United States	`en_US`

Library Support for Devices and iOS versions

The Mobile Express Checkout Library and the demo application fully support iOS 4.0. You can compile the library files and the demo application into the following configurations:

- 3.0, 3.1.x (iPhone only)
- 3.x (Universal)
- 4.x

The library supports armv6 and armv7 architectures for SDK 4.0 and below. It supports only Xcode 3.2.3.

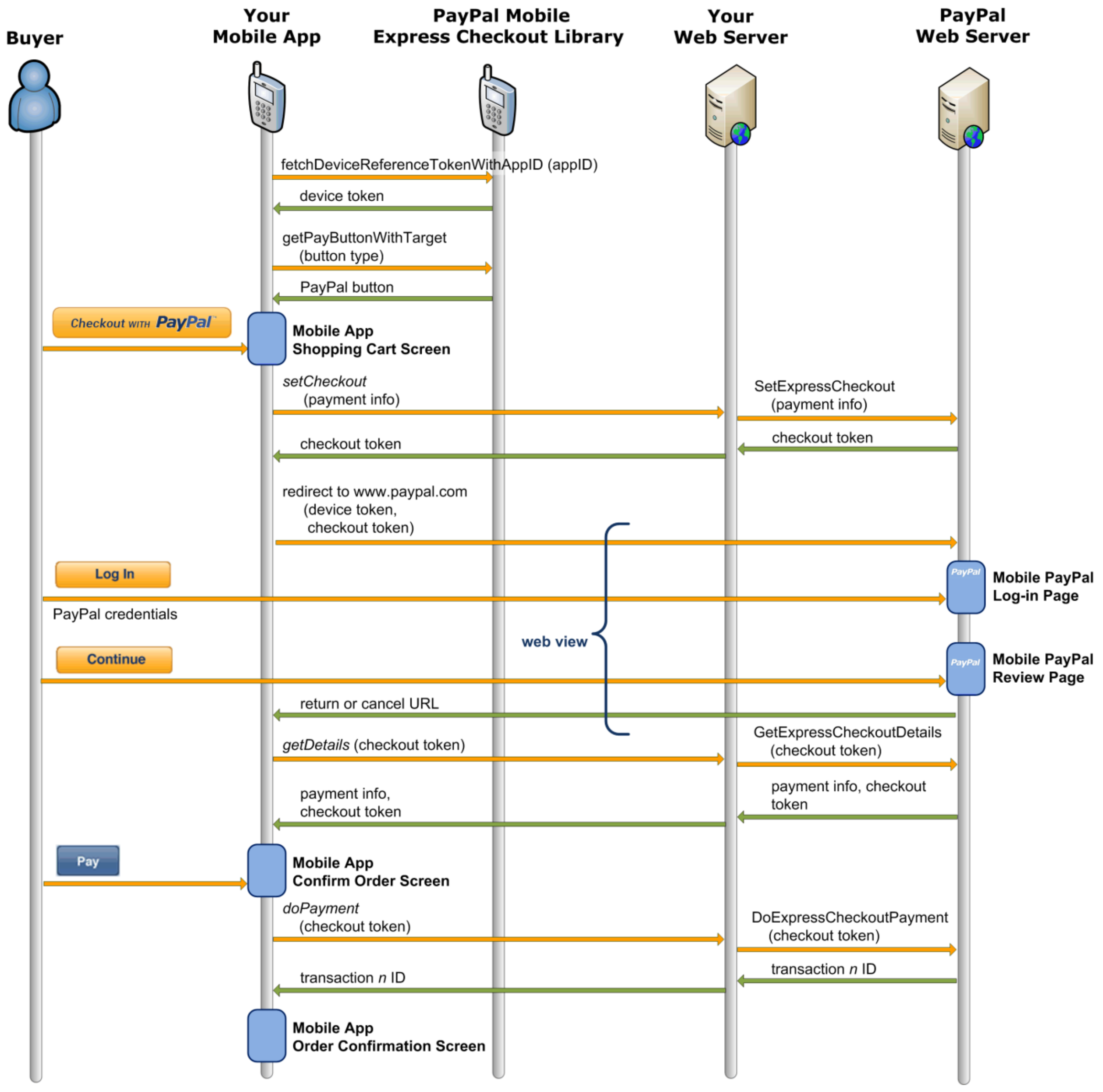
Adding the Mobile Express Checkout Library to Your Xcode Project

PayPal provides a package that contains the header file `PayPal.h` and the static library file `libPayPalEC.a`.

1. Open your Xcode project.
2. CONTROL+CLICK your project, and then select **Add > Existing Files…**.
3. Select the `.h` and `.a` files, and then click **Add**.

Method Sequence with the PayPal Button in Your Mobile App

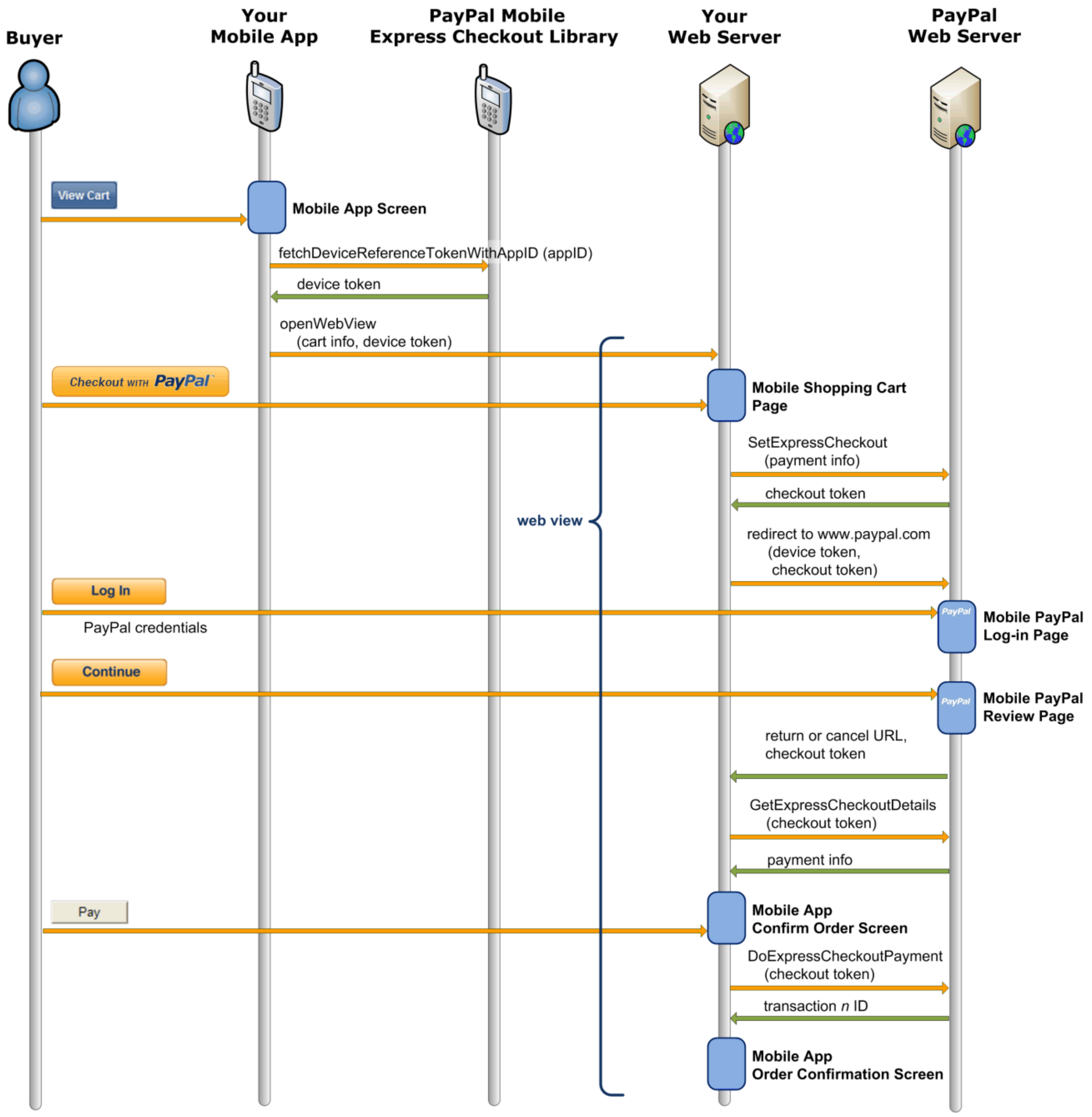
The following diagram illustrates the sequence of methods that embed only the mobile PayPal payment pages in a web view within your mobile application.


Buyer
Your Mobile App
PayPal Mobile Express Checkout Library
Your Web Server
PayPal Web Server
fetchDeviceReferenceTokenWithAppID (appID)
device token
getPayButtonWithTarget (button type)
PayPal button
Checkout WITH PayPal
Mobile App Shopping Cart Screen
setCheckout (payment info)
SetExpressCheckout (payment info)
checkout token
checkout token
redirect to www.paypal.com (device token, checkout token)
Log In
Mobile PayPal Log-in Page
PayPal credentials
web view
Continue
Mobile PayPal Review Page
return or cancel URL
getDetails (checkout token)
GetExpressCheckoutDetails (checkout token)
payment info, checkout token
payment info, checkout token
Pay
Mobile App Confirm Order Screen
doPayment (checkout token)
DoExpressCheckoutPayment (checkout token)
transaction n ID
transaction n ID
Mobile App Order Confirmation Screen

IMPORTANT: Never send Express Checkout requests from your mobile application directly to PayPal. The requests require your PayPal API credentials. Placing your credentials on mobile devices exposes you and PayPal to unacceptable security risks. Send Express Checkout requests only from secure servers.

Method Sequence with the PayPal Button on Your Mobile Webpage

The following diagram illustrates the sequence of methods that embed your entire mobile Express Checkout implementation in a web view within your mobile application.


Buyer
Your Mobile App
PayPal Mobile Express Checkout Library
Your Web Server
PayPal Web Server
View Cart
Mobile App Screen
fetchDeviceReferenceTokenWithAppID (appID)
device token
openWebView (cart info, device token)
Checkout WITH PayPal
Mobile Shopping Cart Page
SetExpressCheckout (payment info)
checkout token
web view
redirect to www.paypal.com (device token, checkout token)
Log In
PayPal credentials
Mobile PayPal Log-in Page
Continue
Mobile PayPal Review Page
return or cancel URL, checkout token
GetExpressCheckoutDetails (checkout token)
payment info
Pay
Mobile App Confirm Order Screen
DoExpressCheckoutPayment (checkout token)
transaction n ID
Mobile App Order Confirmation Screen

Sample Code

The sample code in this section comes from sample application code included with the download of Mobile Express Checkout Library. Get the download of the library from x.com/mobile.

Library Header File

The library header file, `PayPal.h`, includes the following definitions.

```
#import <UIKit/UIKit.h>
...
```

Use the following protocol to implement your own delegate class. Your class receives device tokens from the library when your code attempts to fetch a device token.

```
@protocol DeviceReferenceTokenDelegate <NSObject>

@required
- (void)receivedDeviceReferenceToken:(NSString *)token;
- (void)couldNotFetchDeviceReferenceToken; @end
```

Use the following interface to access features of the Mobile Express Checkout Library.

```
@interface PayPal : NSObject <UIWebViewDelegate> {
      @private
      BOOL initialized; //whether the PayPal object is initialized.
      BOOL paymentsEnabled;
      NSString *appID;
      NSString *lang;
      PayPalEnvironment environment;

      NSString *errorMessage;
      NSMutableArray *payButtons;

      id<DeviceReferenceTokenDelegate> delegate;
}

@property (nonatomic, retain) NSString *lang;
@property (nonatomic, retain) NSString *errorMessage;
@property (nonatomic, retain) NSMutableArray *payButtons;

@property (nonatomic, readonly) NSString *appID;
@property (nonatomic, readonly) BOOL initialized;
@property (nonatomic, readonly) BOOL paymentsEnabled;
@property (nonatomic, readonly) PayPalEnvironment environment;

+ (PayPal*)getInstance;

- (void)fetchDeviceReferenceTokenWithAppID:(NSString const *)inAppID
forEnvironment:(PayPalEnvironment)env
withDelegate:(id<DeviceReferenceTokenDelegate>)del;
- (void)fetchDeviceReferenceTokenWithAppID:(NSString const *)inAppID
withDelegate:(id<DeviceReferenceTokenDelegate>)del;
```

```
- (UIButton *)getPayButtonWithTarget:(NSObject const *)target
andAction:(SEL)action andButtonType:(PayPalButtonType)inButtonType;

@end
```

Fetching the Device Token

Regardless where you place the PayPal button, you must fetch a device token from the library by calling `fetchDeviceReferenceTokenWithAppID()`. If you place the PayPal button in your mobile application, fetch a device token just before you get the PayPal button. If you place the PayPal button on your mobile website, fetch a device token before you open a web view of your mobile checkout pages.

The sample application implements the `fetchDeviceReferenceTokenWithAppID()` method in the same class that serves as the device-token delegate, `Order2ViewController`.

```
if (!tokenFetchAttempted) {

      // Set this switch to FALSE when you initialize the class.
      tokenFetchAttempted = TRUE;

      // Fetch the device token just before you
      // display the page with the PayPal button.
      [[PayPal getInstance] fetchDeviceReferenceTokenWithAppID:@"APP-
yourAppID" withDelegate:self];
            return;
```

The class serves as its own delegate, so it implements the following methods of the `DeviceReferenceTokenDelegate` protocol.

```
- (void)receivedDeviceReferenceToken:(NSString *)token {
      // Stash the device token somewhere to use later.
      [ECNetworkHandler sharedInstance].deviceReferenceToken = token;
}

- (void)couldNotFetchDeviceReferenceToken {
      // Record the errorMessage that tells what went wrong.
      NSLog(@"DEVICE REFERENCE TOKEN ERROR: %@", [PayPal
getInstance].errorMessage);
      [ECNetworkHandler sharedInstance].deviceReferenceToken = @"";
}
```

You mobile application can proceed with a null device token.

Placing the PayPal Button in Your Mobile Application

To place the PayPal button in your mobile application, call the `getPayButtonWithTarget()` method. In the following example, the `payWithPayPal()` method is you own method that the system calls when the buyers click the PayPal button.

The sample application implements the `payWithPayPal()` callback method in the class that gets the button, `ReviewOrderViewController`.

```
UIButton *button = [[PayPal getInstance] getPayButtonWithTarget:self
andAction:@selector(payWithPayPal) andButtonType:BUTTON_278x43];
```

The `getPayButtonWithTarget` method follows standard memory management conventions and is autoreleased.

Redirecting Buyers to PayPal

Whether you place the PayPal button in your mobile application or on your mobile website, use the same redirect to PayPal.

```
https://www.paypal.com/cgi-bin/webscr?cmd=_express-checkout-mobile
&drt=valueFromMobileExpressCheckoutLibrary&token=valueFromSetExpressChe
ckoutResponse
```

Completing the Payment

Whether you place the PayPal button in your mobile application or on your mobile website, complete the payment by sending `GetExpressCheckoutDetails` and `DoExpressCheckoutPayment` requests from your mobile website.

IMPORTANT: Never send Express Checkout requests from your mobile application directly to PayPal. The requests require your PayPal API credentials. Placing your credentials on mobile devices exposes you and PayPal to unacceptable security risks. Send Express Checkout requests only from secure servers.

If you place the PayPal button in your mobile application, monitor the web view for the redirect from PayPal to the `returnURL` or `cancelURL`. Set these URLs in your `SetExpressCheckout` request. Complete the payment only if PayPal redirects the buyer to the `returnURL`.

For more information on the Express Checkout API, see:

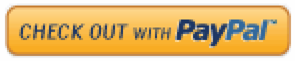
- Express Checkout Integration Guide
- Name-Value Pair API Developer Guide and Reference
- SOAP API Reference

2. The Checkout Experience with the Mobile Express Checkout Library

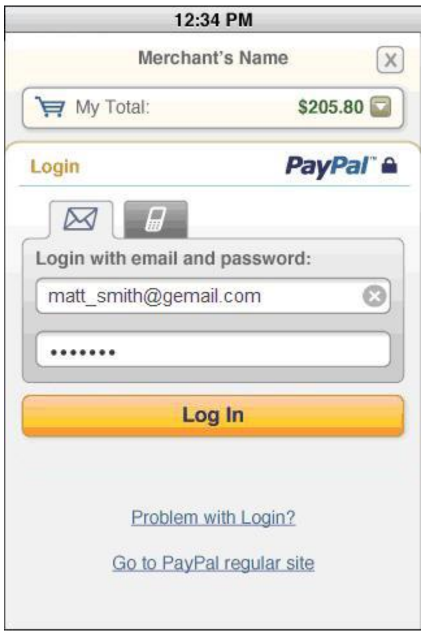
The screen shots that follow illustrate the checkout experience when you embed your mobile implementation of Express Checkout in your mobile application. The embedded checkout experience is the same whether you place the **Checkout with PayPal** button in your mobile application or on your mobile website.

Express Checkout Experience

Your Mobile Screen or Mobile Web Page



Mobile PayPal Log-in Page



12:34 PM

Merchant's Name

My Total: $205.80

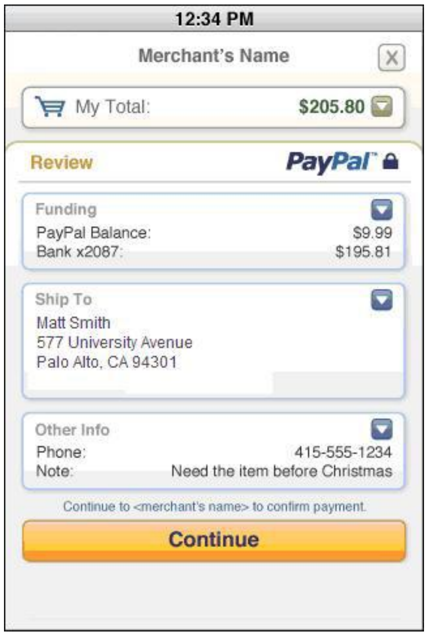
Login

Login with email and password:

matt_smith@gemail.com

•••••••

Mobile PayPal Review Page



12:34 PM

Merchant's Name

My Total: $205.80

Review

Funding

PayPal Balance: $9.99

Bank x2087: $195.81

Ship To

Matt Smith

577 University Avenue

Palo Alto, CA 94301

Other Info

Phone: 415-555-1234

Note: Need the item before Christmas

Continue to <merchant's name> to confirm payment.

After buyers click **Checkout with PayPal,** call `SetExpressCheckout` to begin a PayPal payment. Then, redirect the buyer's browser to PayPal. Include the device token from the library and the checkout token from `SetExpressCheckout` as URL parameters.

PayPal displays the mobile PayPal log-in page when you include a device token with your redirect to PayPal. Buyers enter an email address and password or a mobile phone number and mobile PIN to log in to PayPal.

PayPal displays the mobile PayPal Review page after buyers log in successfully. The mobile Review page fully supports Express Checkout when you include a device token with your redirect to PayPal.

When satisfied with the payment information the Review page, buyers click **Continue** to complete the payment in your mobile application or on your mobile website. If buyers pay on PayPal in your implementation of Express Checkout, the button label on the review page reads, “Pay Now” instead of “Continue.”

3. Submitting Your Application to PayPal

Log in or sign up on PayPal's developer website www.x.com. From there, click the MyApps tab to submit your mobile application. For PayPal to test your application, upload your build signed with the Ad Hoc Distribution profile. PayPal reviews applications within 24 hours and responds by sending you your PayPal Application ID.

After you receive your live Application ID, make sure to change the following items in your demo code (if you choose to use it):

- **Application ID:** in `fetchDeviceReferenceTokenWithAppId`
- **Environment:** in `fetchDeviceReferenceTokenWithAppId`
- **Recipient:** in `PAYMENTREQUEST_`*`n`*`_SELLERPAYPALACCOUNTID` field of the `SetExpressCheckout` request.

A. Creating an Ad Hoc Build

The Ad Hoc distribution method for iPhone apps allows for distribution of the build to internal or external sources. PayPal provides the Device ID values for PayPal's devices. In the process below, you add PayPal's devices to your Ad Hoc Provisioning Profile. You then compile the build, sign it with your Ad Hoc Provisioning Profile, and deliver the zipped build and the Ad Hoc Provisioning Profile to PayPal.

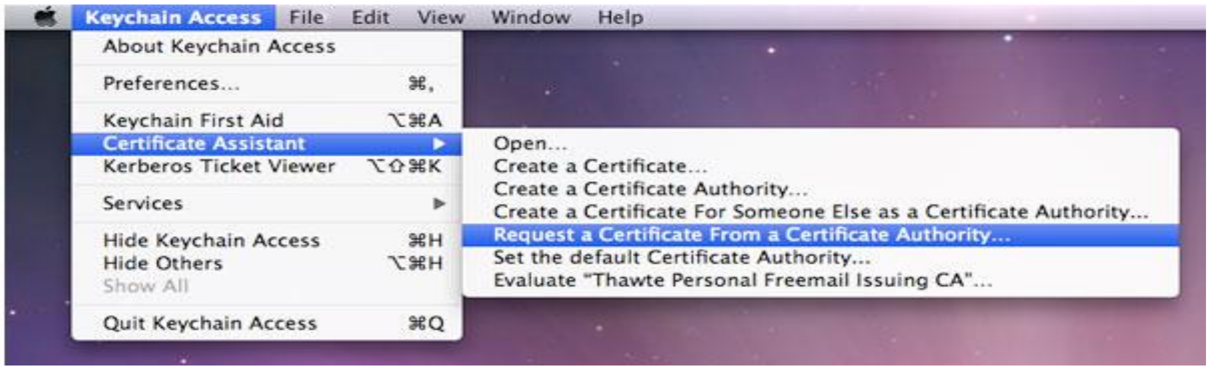
Creating a Distribution Certificate

Distribution Certificates are paired with private keys linked to computers. A Distribution Certificate is one component of the Distribution Provisioning Profile that you use to sign the build in Xcode.

Creating and Approving a Certificate Signing Request

To request an iPhone Development Certificate, generate a Certificate Signing Request (CSR) utilizing the Keychain Access application in Mac OS X Leopard. When you create a CSR, Keychain Access generates your public and private key pair to establish your iPhone Developer identity. Your private key is stored in the login Keychain by default, and you can view it in the Keychain Access application under the 'Keys' category.

1. In your Applications folder, open the Utilities folder and launch Keychain Access.
2. In the Preferences menu, set Online Certificate Status Protocol (OSCP) and Certificate Revocation List (CRL) to "Off".
3. Choose Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority.


Keychain Access
File
Edit
View
Window
Help
About Keychain Access
Preferences... ⌘,
Keychain First Aid ⌥⌘A
Certificate Assistant
Kerberos Ticket Viewer ⌥⇧⌘K
Services
Hide Keychain Access ⌘H
Hide Others ⌥⌘H
Show All
Quit Keychain Access ⌘Q
Open...
Create a Certificate...
Create a Certificate Authority...
Create a Certificate For Someone Else as a Certificate Authority...
Request a Certificate From a Certificate Authority...
Set the default Certificate Authority...
Evaluate "Thawte Personal Freemail Issuing CA"...

NOTE: If you highlight a non-compliant private key in the Keychain during this process, the Program Portal cannot accept the resulting Certificate Request. Make sure that you are select "Request a Certificate From a Certificate Authority...," not "Request a Certificate From a Certificate Authority with <Private Key>…".

4. In the **User Email Address** field, enter your email address.
   Make sure that the email address entered matches the information that you submitted when you registered as an iPhone Developer.
5. In the **Common Name** field, enter your name.
   Make sure that the name you enter matches the information that you submitted to register as an iPhone Developer. A CA (Certificate Authority) Email Address is not required. The 'Required' message disappears after completing the following step.
6. Select the **Saved to Disk** radio button and if prompted, select **Let me specify key pair information** and click the **Continue** button.
7. If you selected **Let me specify key pair**, specify a file name and click the **Save** button.
8. In the screen that follows, select **2048 bits** for the **Key Size** and **RSA** for the **Algorithm**. Then, click the **Continue** button.
9. The Certificate Assistant creates a CSR file on your desktop.

IMPORTANT: Export the private key immediately and share it with all developers who need to compile and sign builds for distribution. For more details, see "Saving the Private Key and Transferring It to Other Systems" on page 29.

Creating a Distribution Certificate

After you create the Certificate Signing Request (CSR), you can create a Distribution Certificate.

1. Log in to the iPhone Developer Program Portal and navigate to **Certificates > Distribution** and click the **Add Certificate** button.
2. Click the **Upload file** button, select your CSR and click 'Submit'.
   If you did not set the Key Size to 2048 bits during the CSR creation process, the Portal cannot accept the CSR.
3. Click the **Approve** button to approve your iPhone Distribution Certificate.
4. In the **Certificates** > **Distribution** section of the Portal, CONTROL+CLICK the **WWDR Intermediate Certificate** link and select **Saved Linked File to Downloads** to initiate download of the certificate.
5. After downloading, double-click the certificate to launch Keychain Access and install.
6. In the same area of the Program Portal, click the name of the iPhone Distribution Certificate to download.
7. On your local machine, double-click the downloaded `.cer` file to launch Keychain Access and install your certificate.

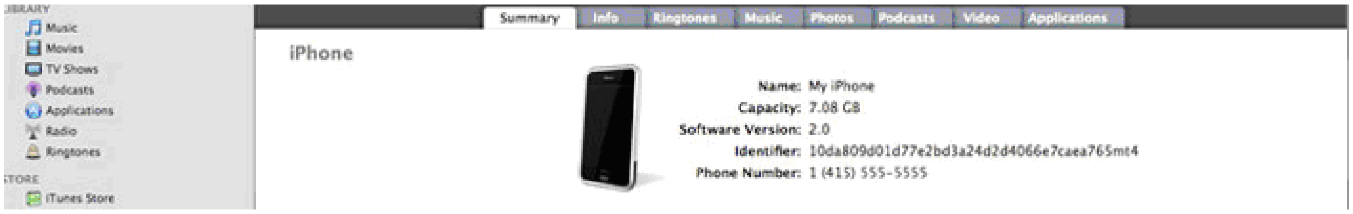
Adding Device IDs

The Devices section of the iPhone Developer Program Portal lets you enter the Apple devices that you use for iOSdevelopment. To install your iOSapplication on an Apple device, enter the Unique Device Identifier (UDID) for each iPhone and iPod touch in the Program Portal. A UDID is a 40-character string that is tied to a single device. UDIDs are similar to hardware serial numbers. These UDIDs are included in the provisioning profiles that you create later. Enter a maximum of 100 devices for your development team.

You need to add your devices, as well as PayPal's devices. You receive UDID values for PayPal's devices from PayPal.

Locating your Device ID

1. Connect your device to your Mac and open Xcode.
2. In Xcode, navigate to the **Window** drop-down menu and select **Organizer**.



Name: My iPhone
Capacity: 7.08 GB
Software Version: 2.0
Identifier: 10da809d01d77e2bd3a24d2d4066e7caea765mt4
Phone Number: 1 (415) 555-5555

The 40-hex character string in the **Identifier** field is your device's UDID.

Adding Devices to the iPhone Developer Program Portal

1. Navigate to the **Devices** section of the Program Portal and click **Add Device**.
2. Enter a descriptive name for the device, as well as the UDID, and then click **Submit**.

Using Updated Provisioning Profiles for New Devices

For new devices to be supported, add them to the Program Portal by following the preceding steps. Then, you must edit the provisioning profile and select the new devices selected for support.

You must import your updated provisioning file into Xcode and use it to sign new builds. You old provisioning profile cannot work for builds signed with the updated profile. Make sure that you distribute the new profile to all Ad Hoc users of the new build, not just users of the devices that you added.

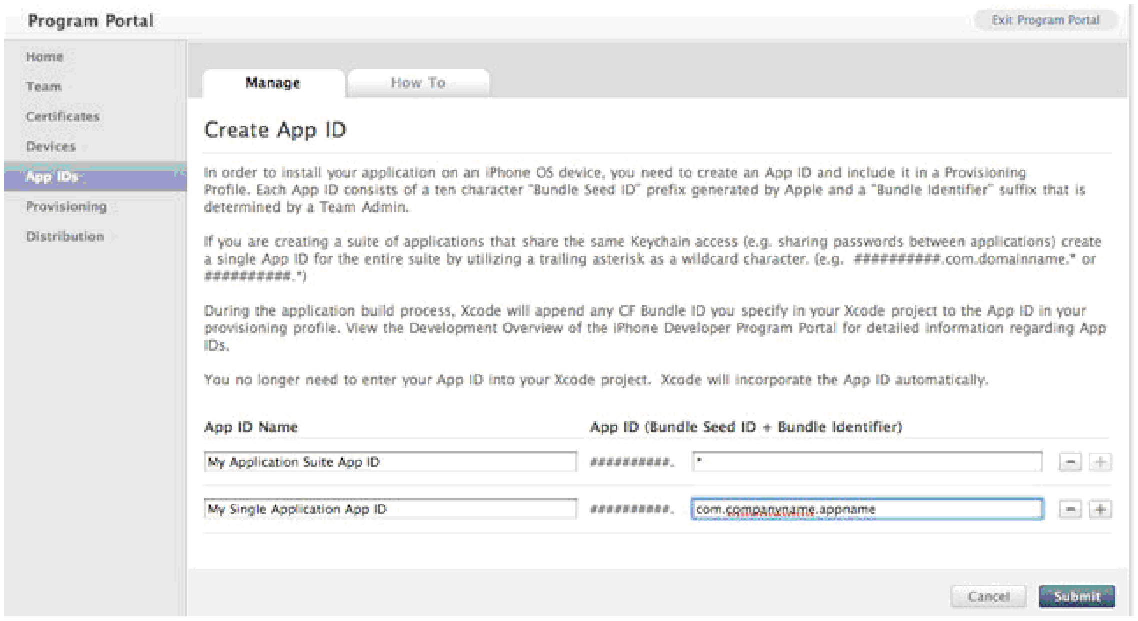
Creating the App ID

An App ID is a unique identifier that iOS uses let your application connect to the Apple Push Notification service, to share keychain data between applications, and to communicate with external hardware accessories that you want paired with your iOS application. To install your application on an iOSdevice, you must create an App ID.

Each App ID consists of a universally unique 10 character "Bundle Seed ID" prefix generated by Apple and a "Bundle Identifier" suffix that is entered by a Team Admin in the Program Portal. PayPal recommends the use of a reverse-domain-name-style string for the "Bundle Identifier" portion of the App ID. An example App ID would be:

```
8E549T7128.com.apple.AddressBook
```

1. Navigate to the **App ID** section of the Program Portal.
2. Click **Add ID**.



Create App ID

In order to install your application on an iPhone OS device, you need to create an App ID and include it in a Provisioning Profile. Each App ID consists of a ten character "Bundle Seed ID" prefix generated by Apple and a "Bundle Identifier" suffix that is determined by a Team Admin.

If you are creating a suite of applications that share the same Keychain access (e.g. sharing passwords between applications) create a single App ID for the entire suite by utilizing a trailing asterisk as a wildcard character. (e.g. ##########.com.domainname.* or ##########.*)

During the application build process, Xcode will append any CF Bundle ID you specify in your Xcode project to the App ID in your provisioning profile. View the Development Overview of the iPhone Developer Program Portal for detailed information regarding App IDs.

You no longer need to enter your App ID into your Xcode project. Xcode will incorporate the App ID automatically.

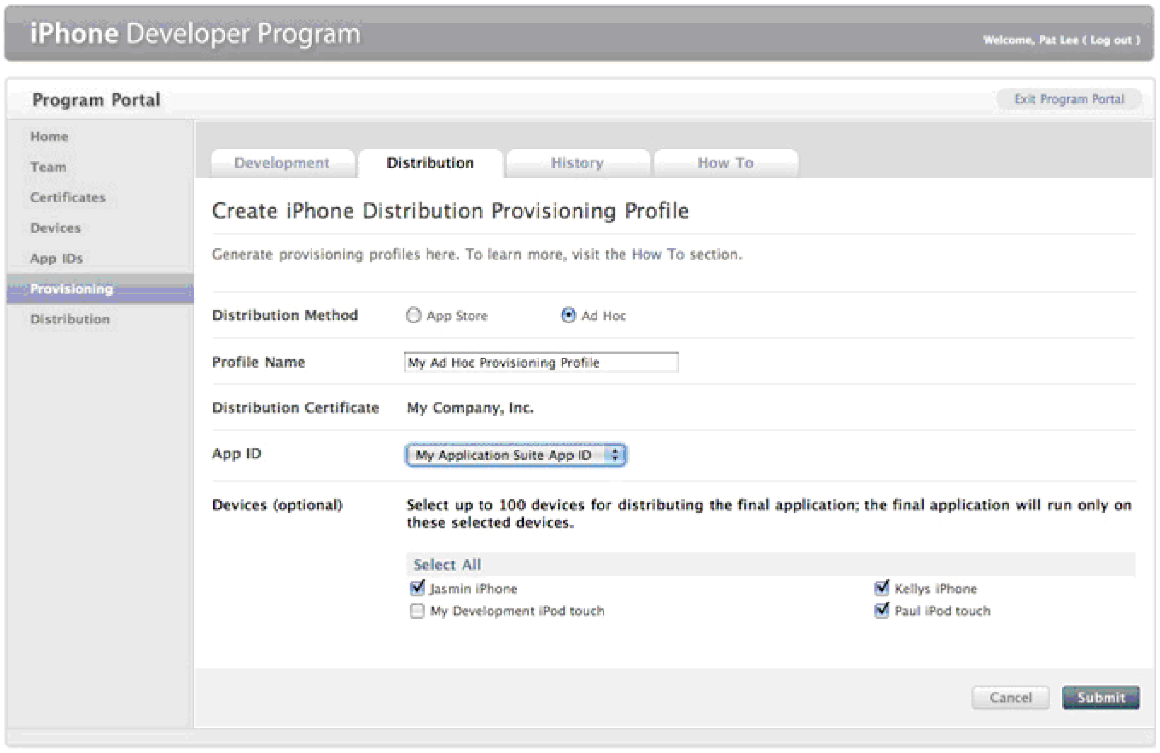
App ID Name	App ID (Bundle Seed ID + Bundle Identifier)	
My Application Suite App ID	##########.	*
My Single Application App ID	##########.	com.companyname.appname

3. Enter a common name for your App ID.
   This is a name for easy reference and identification within the Program Portal.
4. Enter a Bundle Identifier in the free-form text field.
   PayPal recommends a reverse-domain-name-style string, such as com.domainname.applicationname. For applications suites that share the same Keychain access, use a wild-card character in the Bundle Identifier, such as com.domainname.* or *. Your Bundle Identifier must match the CF Bundle Identifier that you use for your application in Xcode.
5. Click **Submit**.
   The 10-character Bundle Seed ID is generated and concatenated with the Bundle Identifier that you entered. The resulting string is your App ID.

Creating a Distribution Provisioning Profile

To successfully build your application in Xcode for Ad Hoc distribution, you must create and download an Ad Hoc Distribution Provisioning Profile.

1. Navigate to the **Provisioning** section of the Program Portal.
2. Click the **Distribution** tab.



Create iPhone Distribution Provisioning Profile

Generate provisioning profiles here. To learn more, visit the How To section.

Distribution Method	App Store [✓ Ad Hoc]
Profile Name	My Ad Hoc Provisioning Profile
Distribution Certificate	My Company, Inc.
App ID	My Application Suite App ID
Devices (optional)	Select up to 100 devices for distributing the final application; the final application will run only on these selected devices.

Select All

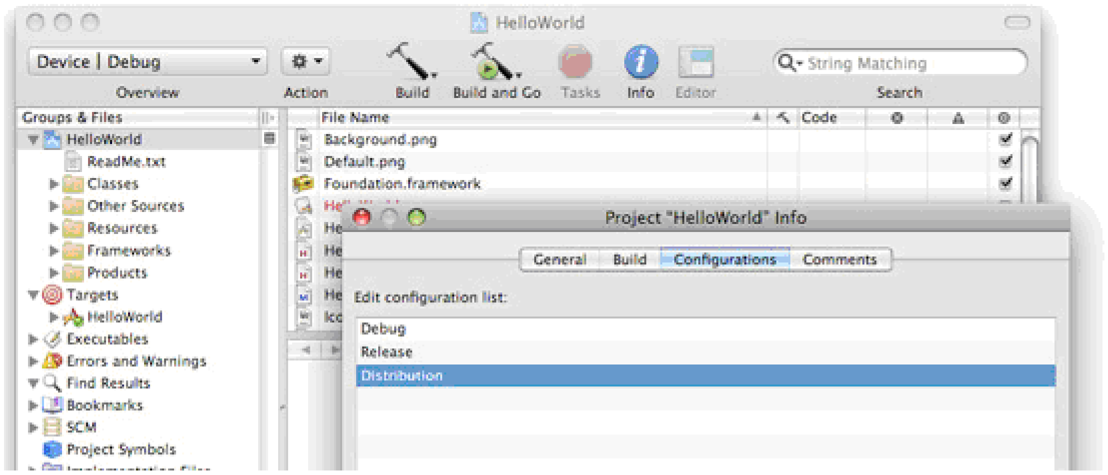
- [✓] Jasmin iPhone
- [ ] My Development iPod touch
- [✓] Kellys iPhone
- [✓] Paul iPod touch

3. Select the **Ad Hoc** radio button.
4. Enter the name for your Ad Hoc Distribution Provisioning Profile.
5. Make sure that your iPhone Distribution Certificate has been created and is displayed.
6. Select the App ID for the application or application suite that you want to distribute.
7. Select up to 100 UDIDs on which you want to run your application.
8. Click the **Submit** button.
9. Download the `.mobileprovision` file by clicking the name of the Distribution Provisioning Profile.
10. Install the `.mobileprovision` file by dragging it onto the Xcode or iTunes icon in the dock.

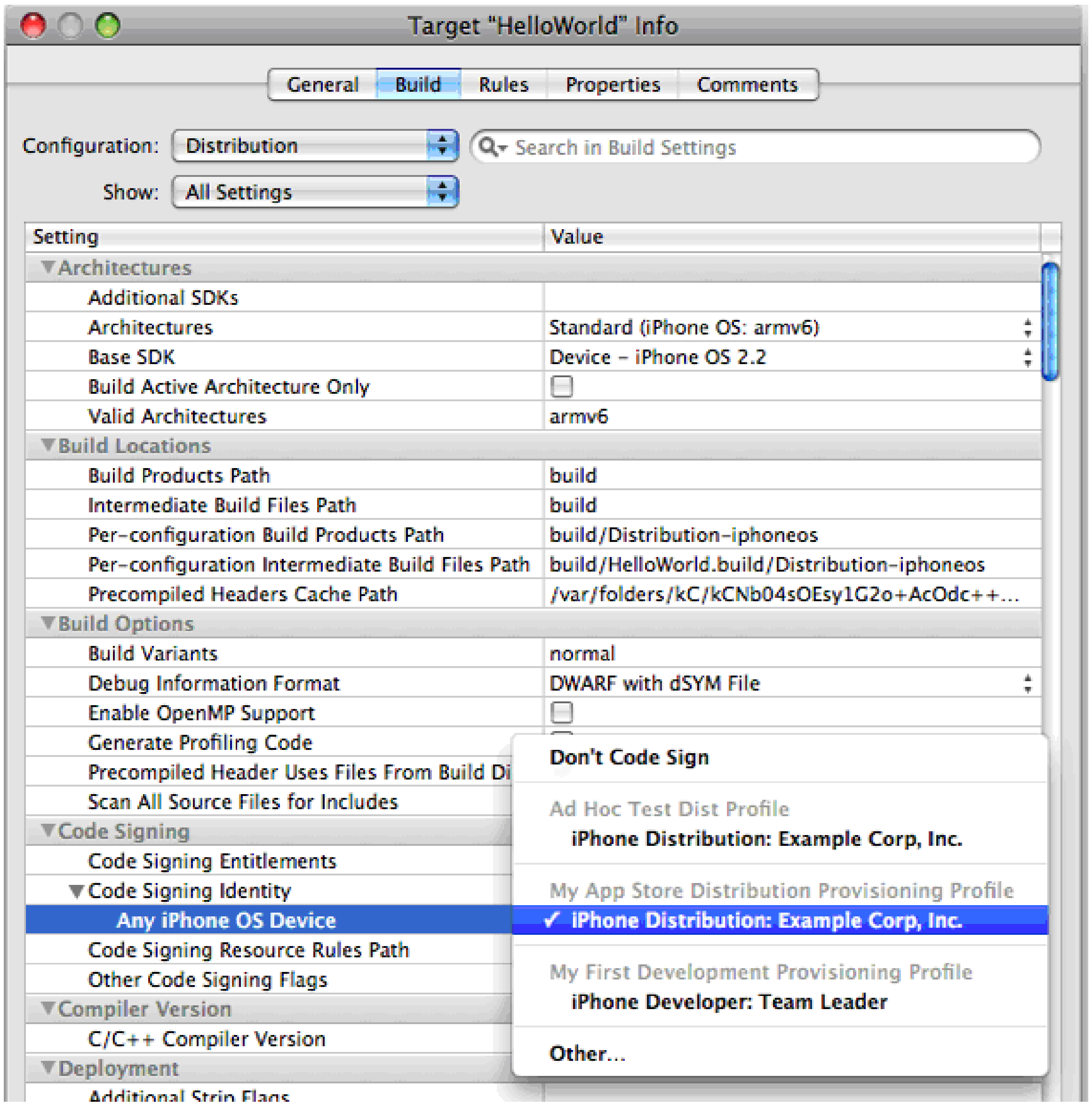
Creating the Build in XCode

After you create the Distribution Provisioning Profile, you can compile and sign your application.

1. Open the project in Xcode and duplicate the **Release** configuration in the Configurations pane of the Info panel for the project.
2. Rename this new configuration "Distribution".



3. In the **Target Info** window, click the **Build** tab and set the **Configuration** to **Distribution**.
4. In the **Target Info** window, navigate to the **Build** pane.
5. Click the **Any iOSDevice** pop-up menu below the **Code Signing Identity** field. Then, select the iPhone Distribution Certificate/Provisioning Profile pair that you want to use to sign and install your code. Your iPhone Distribution certificate is in bold, with its associated Provisioning Profile in gray above it.


Target "HelloWorld" Info

General | Build | Rules | Properties | Comments

Configuration: Distribution

Show: All Settings

Setting	Value
▼Architectures	
Additional SDKs	
Architectures	Standard (iPhone OS: armv6)
Base SDK	Device - iPhone OS 2.2
Build Active Architecture Only	☐
Valid Architectures	armv6
▼Build Locations	
Build Products Path	build
Intermediate Build Files Path	build
Per-configuration Build Products Path	build/Distribution-iphoneos
Per-configuration Intermediate Build Files Path	build/HelloWorld.build/Distribution-iphoneos
Precompiled Headers Cache Path	/var/folders/kC/kCNb04sOEsy1G2o+AcOdc++...
▼Build Options	
Build Variants	normal
Debug Information Format	DWARF with dSYM File
Enable OpenMP Support	☐
Generate Profiling Code	
Precompiled Header Uses Files From Build Di	
Scan All Source Files for Includes	
▼Code Signing	
Code Signing Entitlements	
▼Code Signing Identity	
Any iPhone OS Device	✓ iPhone Distribution: Example Corp, Inc.
Code Signing Resource Rules Path	
Other Code Signing Flags	
▼Compiler Version	
C/C++ Compiler Version	
▼Deployment	
Additional Strip Flags	

Search in Build Settings
Don't Code Sign
Ad Hoc Test Dist Profile
iPhone Distribution: Example Corp, Inc.
My App Store Distribution Provisioning Profile
✓ iPhone Distribution: Example Corp, Inc.
My First Development Provisioning Profile
iPhone Developer: Team Leader
Other...

The preceding example shows 'iPhone Distribution: Example Corp, Inc.' as the Distribution Certificate and 'My App Store Distribution Provisioning Profile' as its associated `.mobileprovision` file.

6. In the **Properties Pane** of the **Target Info** window, enter the Bundle Identifier portion of your App ID. If you used an explicit App ID, enter the Bundle Identifier portion of the App ID in the Identifier field. For example, enter `com.domainname.applicationname` if your App ID is `A1B2C3D4E5.com.domainname.applicationname`. If you used a wildcard asterisk character in your App ID, replace the asterisk with any string.
7. In the project window, select the Distribution Active Configuration from the overview popup and set the Active SDK to the desired Device.
8. In the File Menu, select New File > iOS> Code Signing > Entitlements.

9. Name the file “Entitlements.plist” and click the **Finish** button.
   This creates a copy of the default entitlements file within the project.
10. Select the new Entitlements.plist file, uncheck the **get-task-allow** property, and save the Entitlements.plist file.
11. Select the Target and open the Build settings inspector.
12. In the **Code Signing Entitlements** build setting, type the filename of the new Entitlements.plist file, including the extension.
    Do not specify a path unless you put the Entitlements.plist file somewhere other than the top level of the project.
13. Click the **Build** button.
14. Highlight the app located within the “Products” folder and select **Reveal in Finder** from the Action popup.
15. Use the compress option in Finder to create a `.zip` file that contains your application. Make sure that you compress only the `.app` file only and not the entire build folder.
16. Provide this `.zip` file to PayPal, along with the Distribution Provisioning Profile (`.mobileprovision`) file. Upload this file to x.com as an attachment when you submit your application.

Notes

Saving the Private Key and Transferring It to Other Systems

Make sure that you save your private key somewhere for safekeeping. For example, you may need to develop on multiple computers or you might decide to reinstall your system OS. Without your private key, you cannot sign binaries in Xcode nor test your application on Apple devices.

When a CSR is generated, the Keychain Access application creates a private key on your login keychain. This private key is tied to your user account and cannot be reproduced if lost. If you plan to develop and test on multiple systems, you must import your private key to all of systems on which you work.

1. Open up the Keychain Access Application and select the **Keys** category.
2. CONTROL+CLICK the private key associated with your iPhone Development Certificate, and then click **Export Items** in the menu.
   You can identify the private key by the iPhone Developer: <First Name> <Last Name> public certificate that is paired with it.
3. Save your key in the Personal Information Exchange (`.p12`) file format.
4. When prompted, create a password to use when you import this key on another computer.

Result:

You can transfer this `.p12` file between systems by double-clicking the `.p12` file to install it on a system. When prompted, use the password that you entered in the preceding Step 4.

Verifying a Successful Ad Hoc Distribution Build

1. Open the Build Log detail view and check the presence of the "`embedded.mobileprovision`" file.

   This takes you to the line in the build log that shows the provisioning profile was called successfully. Make sure that the `embedded.mobileprovision` file is located in the proper "Distribution" build directory and not a "Debug" or "Release" build directory. Also, make sure the destination path at the end of the build message is the app that you are building.

2. Search for the term "CodeSign" in the Build Log detail view.

   This takes you to the line in the build log that confirms your application was signed by your iPhone Certificate.

Correcting an Unsuccessful Ad Hoc Distribution Build

Distribution builds can fail if your project is lacking the embedded.mobileprovision file or points to the wrong directory.

1. Select the Target and open the Build Settings Inspector.

   Make sure that you are in the Distribution Configuration.

2. Delete the Code Signing Identity: iPhone Distribution: COMPANYNAME
3. In the Xcode Build Menu, select **Clean all Targets**.
4. Delete any existing build directories in your Xcode project by using Finder.
5. Launch Xcode again and open your Project.
6. Re-enter the code-signing identity iPhone Distribution: COMPANYNAME in the Target Build Settings Inspector.
7. Rebuild your project.