Technical Note TN2265

Troubleshooting Push Notifications

Describes techniques you can use to resolve issues with sending and receiving push (remote) notifications in iOS and Mac OS X.

Introduction

Many iOS and Mac OS X applications present dynamic content delivered over the Internet. Push notifications (also known as remote notifications) are a way to let users know that new or updated content they're interested in is available.

This technote describes techniques you can use to resolve issues with sending and receiving push notifications.

> **Note:** Push notifications are available on Mac OS X starting with Mac OS X Lion.

Back to Top

Issues with Receiving Push Notifications

Registering for Push Notifications

In order to receive push notifications, an application must first register with the Apple Push Notification service ("push service"). Registration has three stages:

1. The application calls the `registerForRemoteNotificationTypes:` method of `UIApplication` (on iOS) or `NSApplication` (on Mac OS X).
2. The application implements the `application:didRegisterForRemoteNotificationsWithDeviceToken:` method of `UIApplicationDelegate` (iOS) or `NSApplicationDelegate` (Mac OS X) to receive the unique device token generated by the push service.
3. The application implements the `application:didFailToRegisterForRemoteNotificationsWithError:` method of `UIApplicationDelegate` (iOS) or `NSApplicationDelegate` (Mac OS X) to receive an error if the registration failed.

The application passes the device token to your provider as a non-object, binary value. The provider is your server that delivers dynamic content to your application.

Registering for push notifications is straightforward, but there are a few important details you need to be aware of.

No Delegate Callbacks

When the first push-capable app is installed, iOS or Mac OS X attempts to establish a persistent network connection to the push service that will be shared by all push-capable apps on the system. If neither delegate callback `application:didRegisterForRemoteNotificationsWithDeviceToken:` nor `application:didFailToRegisterForRemoteNotificationsWithError:` is called, that means that this connection has not yet been established.

This is not necessarily an error condition. The system may not have Internet connectivity at all because it is out of range of any cell towers or Wi-Fi access points, or it may be in airplane mode. Instead of treating this as an error, your app should continue normally, disabling only that functionality that relies on push notifications.

Keep in mind that network availability can change frequently. Once the persistent connection to the push service succeeds, one of the previously-mentioned application delegate methods will be called.

On iOS, push notifications use the cellular data network whenever possible, even if the device is currently using Wi-Fi for other network activity such as web browsing or email. However, the push service will fall back to Wi-Fi if cellular data service isn't available.

If your iOS device is capable of using the cellular data network, check that it has an active cellular data plan. Turn off Wi-Fi in Settings and see if you can still browse the web with Safari, for example. On the other hand, if the push service is using Wi-Fi, any firewalls between your device or computer and the Internet must allow TCP traffic to and from port 5223.

> **Note:** There is a separate persistent connection to the push service for each environment. The operating system establishes a persistent connection to the sandbox environment for development builds; ad hoc and distribution builds connect to the production environment.

Error Delegate Callback

The first time a push-capable iOS application is run, iOS asks the user if they want to receive push notifications for this app. If the app isn't even offering the option of receiving push notifications, the chances are that it is missing its `aps-environment` code signing entitlement. This entitlement is taken from the provisioning profile used when building the app, and it controls which push environment the app will connect to in order to receive remote notifications.

If an iOS app is running on a device, or if the app is a Mac OS X app, the error method `application:didFailToRegisterForRemoteNotificationsWithError:` will be called if the code signing entitlements for accessing the push service are invalid. If you have not implemented this method in your application, you should do that as a way of checking that your `aps-environment` (iOS) or `com.apple.developer.aps-environment` (Mac OS X) entitlement is present at runtime.

You can check your app's code signing entitlements with the codesign tool. The value of the `aps-environment` or `com.apple.developer.aps-environment` entitlement should be either `development` or `production`. Run this command in Terminal pointing at your App Store distribution build:

```
$ codesign -d --entitlements - <YourAppName>.app
```

Check if there is an `aps-environment` or `com.apple.developer.aps-environment` code signing entitlement listed under `Internal requirements`. It would normally appear after the `application-identifier` (iOS) or `com.apple.application-identifier` (Mac OS X) entitlement.

Note: This is a useful test to perform on your distribution builds before submitting them to the App Store. The Application Loader or Xcode do not check if the push entitlement is missing.

If the `aps-environment` or `com.apple.developer.aps-environment` entitlement is missing, you probably created your distribution provisioning profile before you configured your App ID for push notifications using the iOS Provisioning Portal or the Mac Developer Certificate Utility. If your App ID has already been configured for push notifications, go into the Portal or the Developer Certificate Utility and download a new provisioning profile. It most likely will have the desired code signing entitlements in place. You can look at the profile with a text editor like TextEdit and search for `Entitlements`.

Registration Succeeded But No Notifications Received

If your app has registered with the push service but it is not receiving notifications, the problem can be either on the device/computer side or on the provider side. Here are a few things to check on the device/computer side.

The device or computer may have lost its persistent connection to the push service and can't reconnect. Try quitting the app and relaunching it to see if registration completes the next time. (On iOS 4 and later on devices that support multitasking, you will need to force quit the app from the recents list.) If the registration does not complete, iOS or Mac OS X has been unable to re-establish the persistent connection. You can troubleshoot this as described in the previous two sections.

Your app may have sent an incorrect device token to your provider. Your app should always ask for the device token by registering with the push service each time it is launched. Don't store a device token from your app and try to reuse it, because the token can change. Your provider should then pass that same token on to the push service.

Note: The push service knows which app is supposed to receive a notification based on the User ID attribute in your TLS/SSL certificate. iOS or Mac OS X will pass the notification to the app with the

`CFBundleIdentifier` matching this attribute.

Some Notifications Received, but Not All

If you are sending multiple notifications to the same device or computer within a short period of time, the push service will send only the last one.

Here's why. The device or computer acknowledges receipt of each notification. Until the push service receives that acknowledgment, it can only assume that the device or computer has gone off-line for some reason and stores the notification in the quality of service (QoS) queue for future redelivery. The round-trip network latency here is of course a major factor.

As described in the Local and Push Notification Programming Guide, the QoS queue holds a single notification per app per device or computer. If the service receives another notification before the one in the queue is sent, the new notification overwrites the previous one.

All of this points out that the intent is that a notification indicates to an app that something of interest has changed on the provider, and the app should check in with the provider to get the details. Notifications should not contain data which isn't also available elsewhere, and they should also not be stateful.

Observing Push Status Messages

If these tips don't resolve the issue, you can enable additional messages from the APNs daemon on the device or computer.

Enabling Push Status Messages on iOS

To enable logging on iOS, install the configuration profile PersistentConnectionLogging.mobileconfig on your device by either putting the file on a web server and downloading it using Safari on your device, or by sending it as an email attachment and opening the attachment in Mail on your device. You'll be prompted to install "APS/PC Logging".

Once the configuration profile has been installed, turn the device off completely and then turn it back on. Then exercise your app again while watching the console using the Xcode Organizer. Or, sync your device with iTunes and the log will be saved in `~/Library/Logs/CrashReporter/MobileDevice/<DEVICE_NAME>/PersistentConnection/`.

Look for messages from the `apsd` process. Ideally you'll see `Connected to courier x-courier.sandbox.push.apple.com` where `x` is a small integer. That indicates that the device has successfully established its persistent connection to the push service.

You might on the other hand see `Disconnecting in response to connection failure.` That means that the persistent connection failed. In that case, the goal is to figure out what's going on with your network that's causing the connection failure. Check that no firewalls are blocking TCP traffic on port 5223.

The message `Received message for enabled topic <your app's CFBundleIdentifier>` means that the device received a notification from the push service.

On Mac OS X, some log messages refer to the hash of your app's `CFBundleIdentifier`. Here's how to compute this SHA-1 hash.

Listing 1: Computing the Hash of a CFBundleIdentifier

```
$ echo -n "<your CFBundleIdentifier>" | openssl dgst -sha1
```

If things appear to look normal in the log but you're still not receiving notifications, try turning off the Notifications switch in Settings, and then turn it back on. That will try to re-establish the device's persistent connection with APNs.

Enabling Push Status Messages on Mac OS X

To enable logging on Mac OS X, use the commands shown in Listing 2.

Listing 2: Enabling Push Status Messages on Mac OS X.

```
$ sudo defaults write /Library/Preferences/com.apple.applepushserviced APSWriteLogs -bool
TRUE

$ sudo defaults write /Library/Preferences/com.apple.applepushserviced APSLogLevel -int 7

$ sudo killall applepushserviced
```

The logs are stored in `/Library/Logs/applepushserviced.log`.

Back to Top

Issues with Sending Push Notifications

If your app is not receiving push notifications, and everything looks correct on the device or computer, here are some things to check on the provider side.

Problems Connecting to the Push Service

One possibility is that your provider is unable to connect to the push service. This can mean that you don't have the certificate chain needed for TLS/SSL to validate the connection to the service. In addition to the SSL identity (certificate and associated private key) created by the iOS Provisioning Portal or the Mac Developer Certificate Utility, you should also install the Entrust CA (2048) root certificate on your provider. This allows TLS/SSL to verify the full APNs server cert chain. If you need to get this root certificate, you can download it from Entrust's site. Also verify that these identities are installed in the correct location for your provider and that your provider has permission to read them.

You can test the TLS/SSL handshake using the OpenSSL s_client command, like this:

```
$ openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert
YourSSLCertAndPrivateKey.pem -debug -showcerts -CAfile server-ca-cert.pem
```

where `server-ca-cert.pem` is the Entrust CA (2048) root certificate.

Be sure the SSL identity and the hostname are the correct ones for the push environment you're testing. You can configure your App ID in the iOS Provisioning Portal or the Mac Developer Certificate Utility separately for the sandbox and production environment, and you will be issued a separate identity for each environment.

Using the sandbox SSL identity to try to connect to the production environment will return an error like

this:

```
CRITICAL | 14:48:40.304061 | Exception creating ssl connection to Apple: [Errno 1]
_ssl.c:480: error:14094414:SSL routines:SSL3_READ_BYTES:sslv3 alert certificate
revoked
```

Note: The exact error will depend on your particular operating environment.

Handling Malformed Notifications

The simple binary interface drops the connection if the push service receives a notification that is incorrect in some way. Your provider may see this as an `EPIPE` or broken pipe error in response to sending a notification. On the other hand, the enhanced binary interface will send an error response with more detailed information about what was wrong with the notification before dropping the connection. Be sure your provider catches and handles these conditions properly.

The most common problem is an invalid device token. If the token came from the sandbox environment, such as when you are testing a development build in house, you can't send it to the production push service. Each push environment will issue a different token for the same device or computer. If you do send a device token to the wrong environment, the push service will see that as an invalid token and discard the notification.

Note: It is recommended that you run a separate instance of your provider for each push environment to avoid the problem of sending device tokens to the wrong environment.

An invalid device token can also mean that the user has deleted your app from their device or computer. You should check the feedback service at least once a day for device tokens that are no longer valid.

Other possible issues might be sending a payload longer than 256 bytes, your payload might not be formatted correctly, or perhaps your JSON dictionary has incorrect syntax.

An occasional disconnect while your provider is idle is nothing to be concerned about; just re-establish the connection and carry on. If one of the push servers is down, the load balancing mechanism will transparently direct your new connection to another server assuming you connect by hostname and not by static IP address.

Using the Enhanced Binary Interface

If your provider is using the original simple binary interface, consider migrating to the enhanced binary interface. That interface provides more control over the expiration of notifications and also returns more detailed error responses if a problem occurs with a particular notification.

You can read more about the enhanced binary interface in the Local and Push Notification Programming Guide.

Issues with Using the Feedback Service

If you remove your app from your device or computer and then send a push notification to it, you would expect to have the device token rejected, and the invalidated device token should appear on the feedback service. However, if this was the last push-enabled app on the device or computer, it will not show up in the feedback service. This is because deleting the last app tears down the persistent connection to the

push service before the notice of the deletion can be sent.

You can work around this by leaving at least one push-enabled app on the device or computer in order to keep the persistent connection up. To keep the persistent connection to the production environment up, just install any free push-enabled app from the App Store and you should then be able to delete your app and see it appear in the feedback service.

Recall that each push environment has its own persistent connection. So to keep the persistent connection to the sandbox environment up, install another development push-enabled app.

Back to Top

Other Tips and Tricks

IP Address Range Used by the Push Service

Push providers, iOS devices, and Mac computers are often behind firewalls. To send notifications, you will need to have TCP port 2195 open. To reach the feedback service, you will need to have TCP port 2196 open. Devices and computers connecting to the push service over Wi-Fi will need to have TCP port 5223 open.

The IP address range for the push service is subject to change; the expectation is that providers will connect by hostname rather than IP address. The push service uses a load balancing scheme that yields a different IP address for the same hostname. However, the entire 17.0.0.0/8 address block is assigned to Apple, so you can specify that range in your firewall rules.

Resetting the Push Notifications Permissions Alert on iOS

The first time a push-enabled app registers for push notifications, iOS asks the user if they wish to receive notifications for that app. Once the user has responded to this alert it is not presented again unless the device is restored or the app has been uninstalled for at least a day.

If you want to simulate a first-time run of your app, you can leave the app uninstalled for a day. You can achieve the latter without actually waiting a day by setting the system clock forward a day or more, turning the device off completely, then turning the device back on.

Viewing a Certificate Signing Request (CSR)

Part of configuring your App ID for push notifications is creating a certificate signing request, or CSR. Occasionally it's useful to look at the contents of a CSR, which you can do with the OpenSSL req command:

```
$ openssl req -noout -text -in server.csr
```

Back to Top

References

Local and Push Notification Programming Guide

About Apple Push Notification Service

Back to Top

Downloadables

- PersistentConnectionLogging.mobileconfig ("tn2265_PersistentConnectionLogging.zip", 3.0K)

Back to Top

Document Revision History

Date	Notes
2011-09-26	Update link to Entrust root certificate. Replace apsd logging configuration profile with a signed version.
2011-06-08	Added information about push notifications in Mac OS X Lion.
2010-09-29	New document that describes steps developers can take to troubleshoot sending and receiving of push notifications.