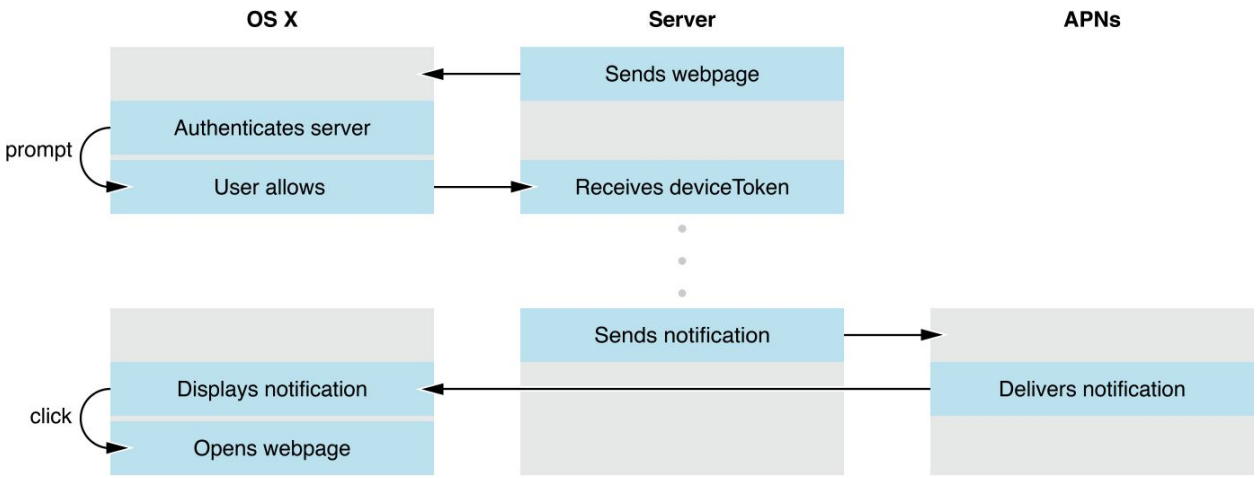
Configuring Safari Push Notifications

1. Introduction

In OS X v10.9 and later, you can dispatch Safari Push Notifications from your web server directly to OS X users by using the Apple Push Notification service (APNs). Not to be confused with local notifications, push notifications can reach your users regardless of whether your website or Safari is open.



To integrate push notifications in your website, you first present an interface that allows the user to opt in to receive notifications. If the user consents, Safari contacts your website requesting its credentials in the form of a file called a *push package*. The push package also contains notification assets used throughout OS X and data used to communicate to a web service you configure. If the push package is valid, you receive a unique identifier for the user on the device known as a *device token*. The user receives the notification when you send the combination of this device token and your message, or *payload*, to APNs.

2. Setup

This guide will hopefully help you in the process management and maintain a coherent way of getting your service implemented. What will be covered in small coherent steps:

1. Generating your website push certificate. An Apple provided certificate from your developer portal is needed in order to send notifications.
   Obtaining the Apple WWDRCA certificate.
   Yes, another certificate is needed when actually sending notifications which and also while creating the push package which is not mentioned in Apple's official documentation.

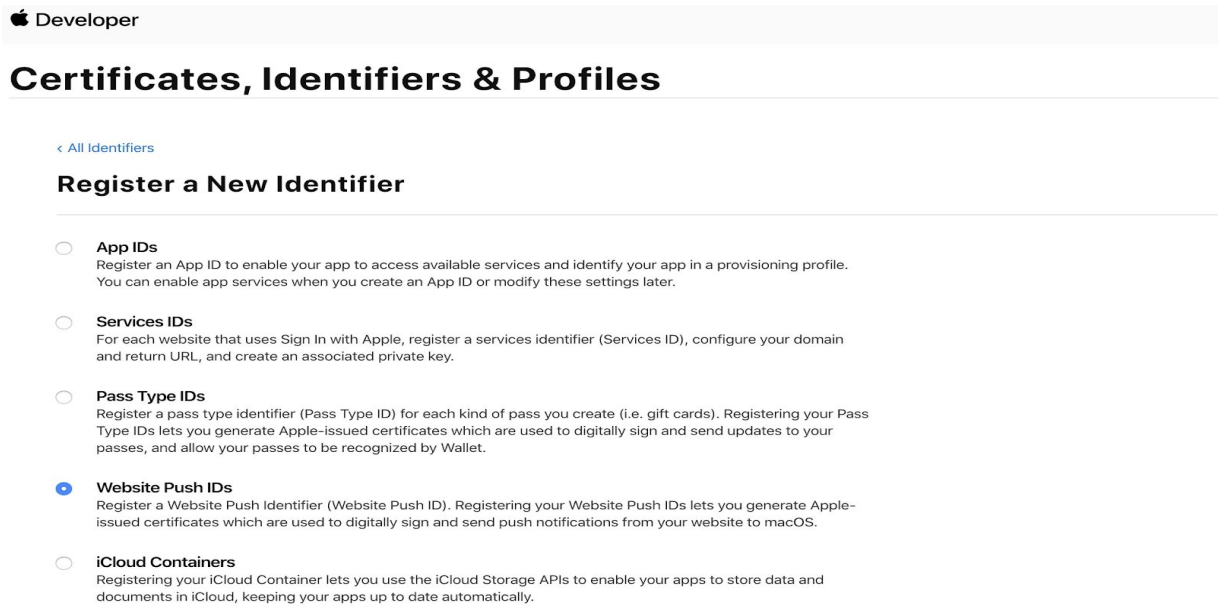
2. Configuring your raw push package folder, and then using the companion PHP file(attached) for your ease to zip up your final push package. We will also discuss the issues one can face while preparing the push package.
   Your "push package" is a folder of icons and configurations that is used by each Mac that subscribes to your notification service.

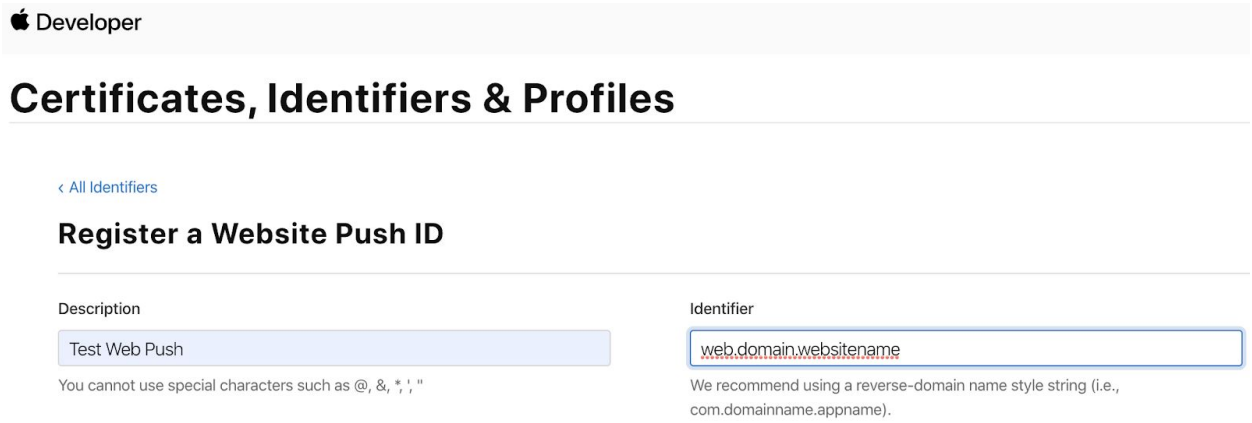
3. Running the front end Javascript to subscribe to your notifications, and getting the device token returned.
   Granting a client access to your notifications is very simple using Javascript. If a valid push package is returned, access will be granted and a device token is given in a JSON response.

● Generating Push Certificates

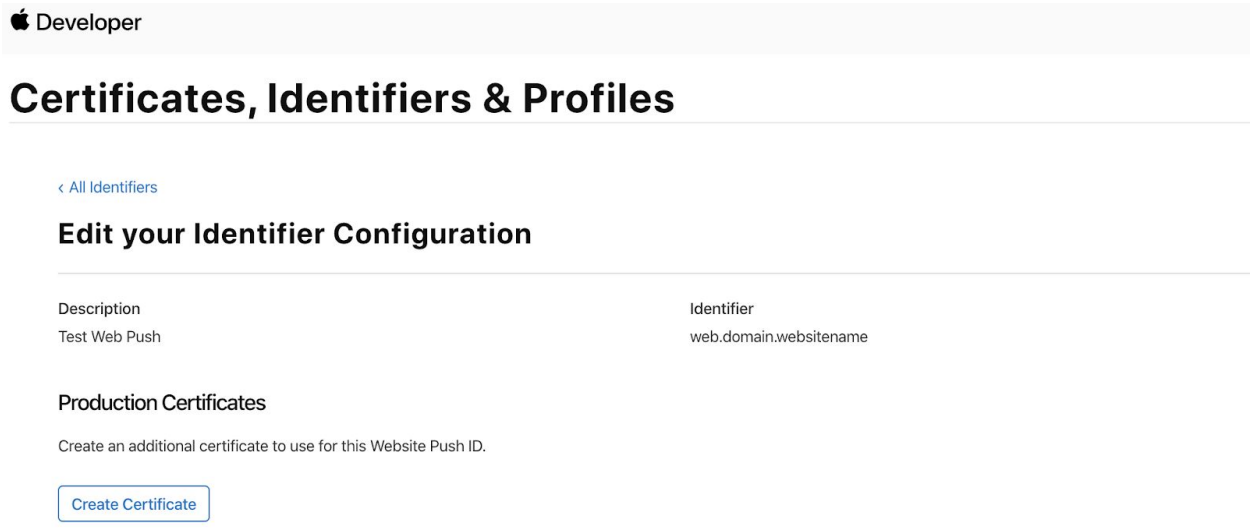
You are required to register in the Certificates, Identifiers & Profiles section of your developer account to send push notifications. Registration requires an Apple developer license.

- **Identifier**. This is your unique reverse-domain string, such as `web.com.example.domain` (the string must start with `web.`). This is also known as the Website Push ID.
- **Website Push ID Description**. This is the name used throughout the Provisioning Portal to refer to your website. Use it for your own benefit to label your Website Push IDs into a more human-readable format.

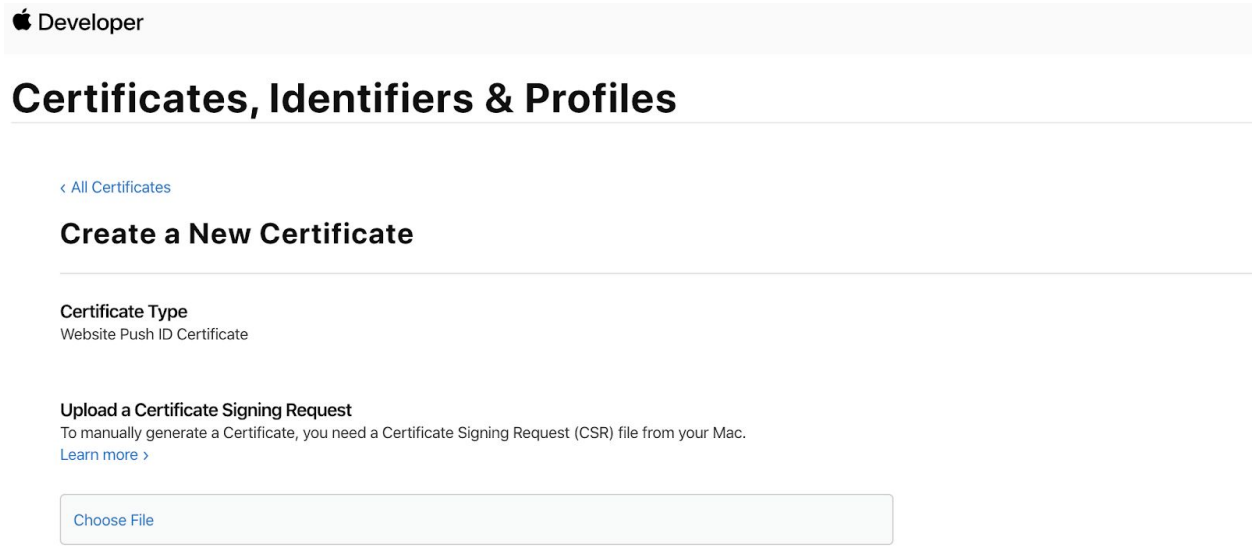




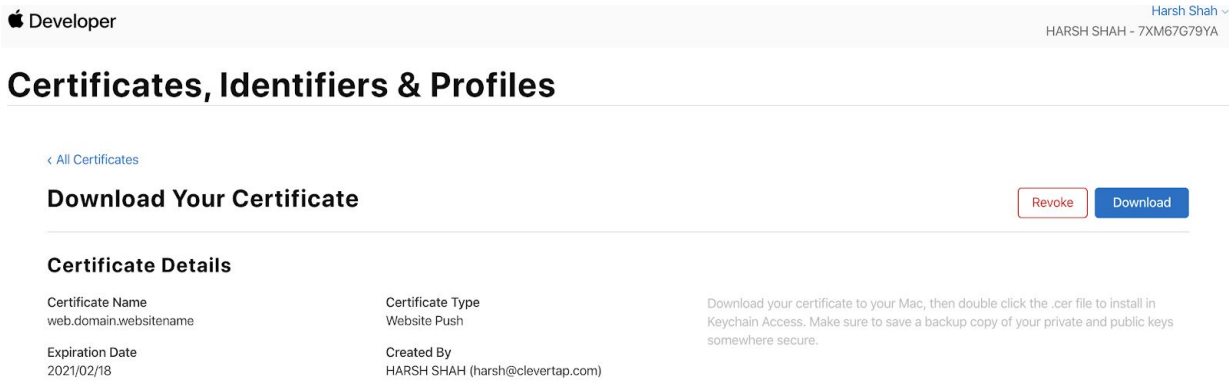
Once the push ID is registered, create a production certificate of that push ID by clicking "*Create Certificate*" under "*Edit your Identifier Configuration*".



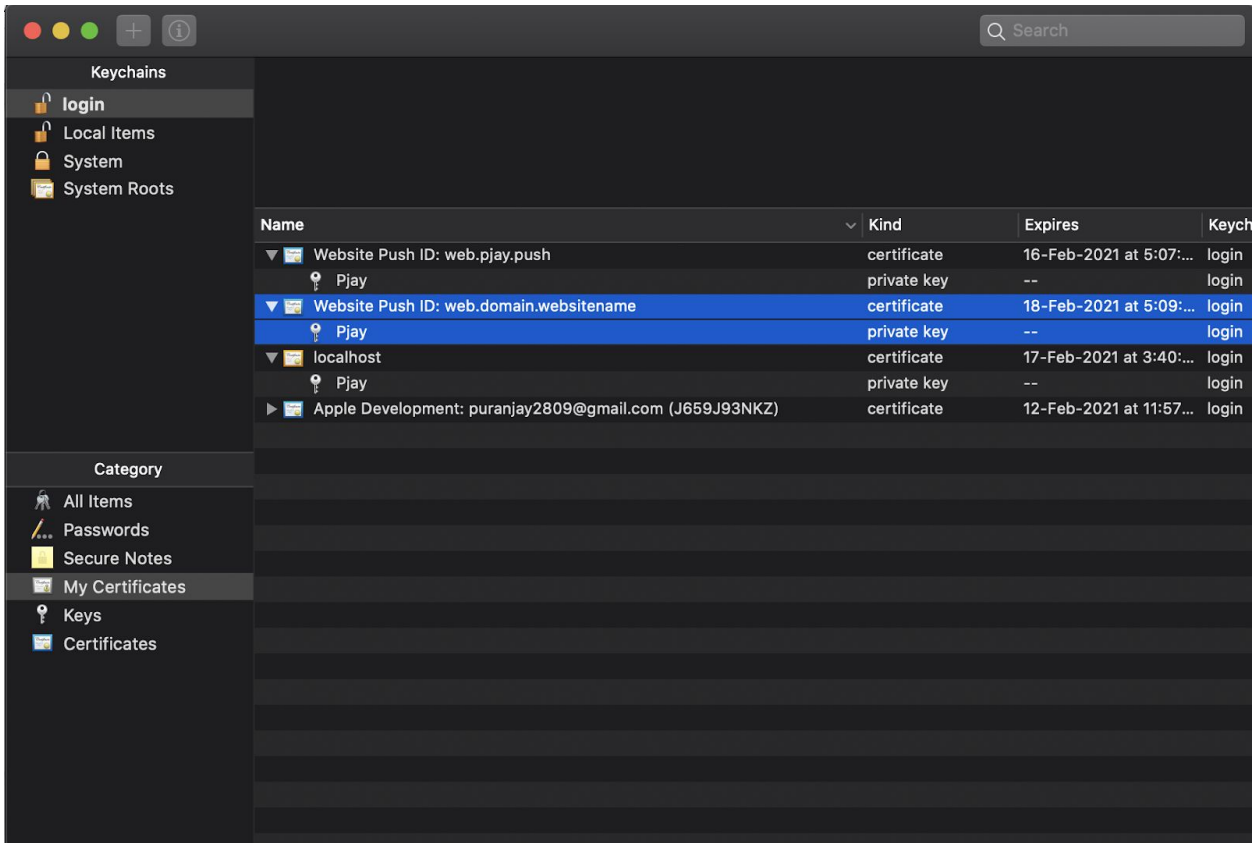
If you already have the Certificate Signing Request, then upload the file from your local machine. If you don't then please click "*Learn more*" to know how to create one. Once created, you can upload the same.

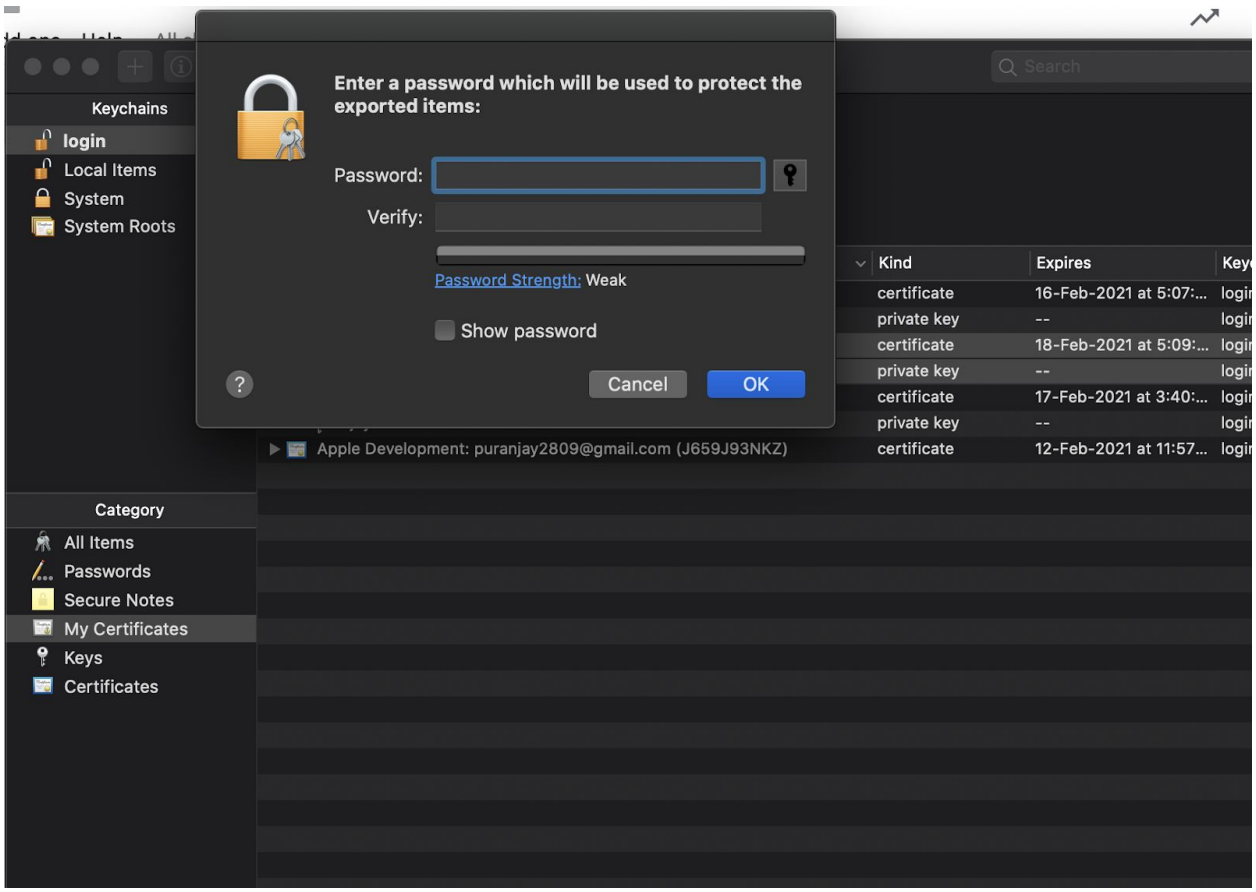


Once CSR is uploaded, you can download the push certificate.



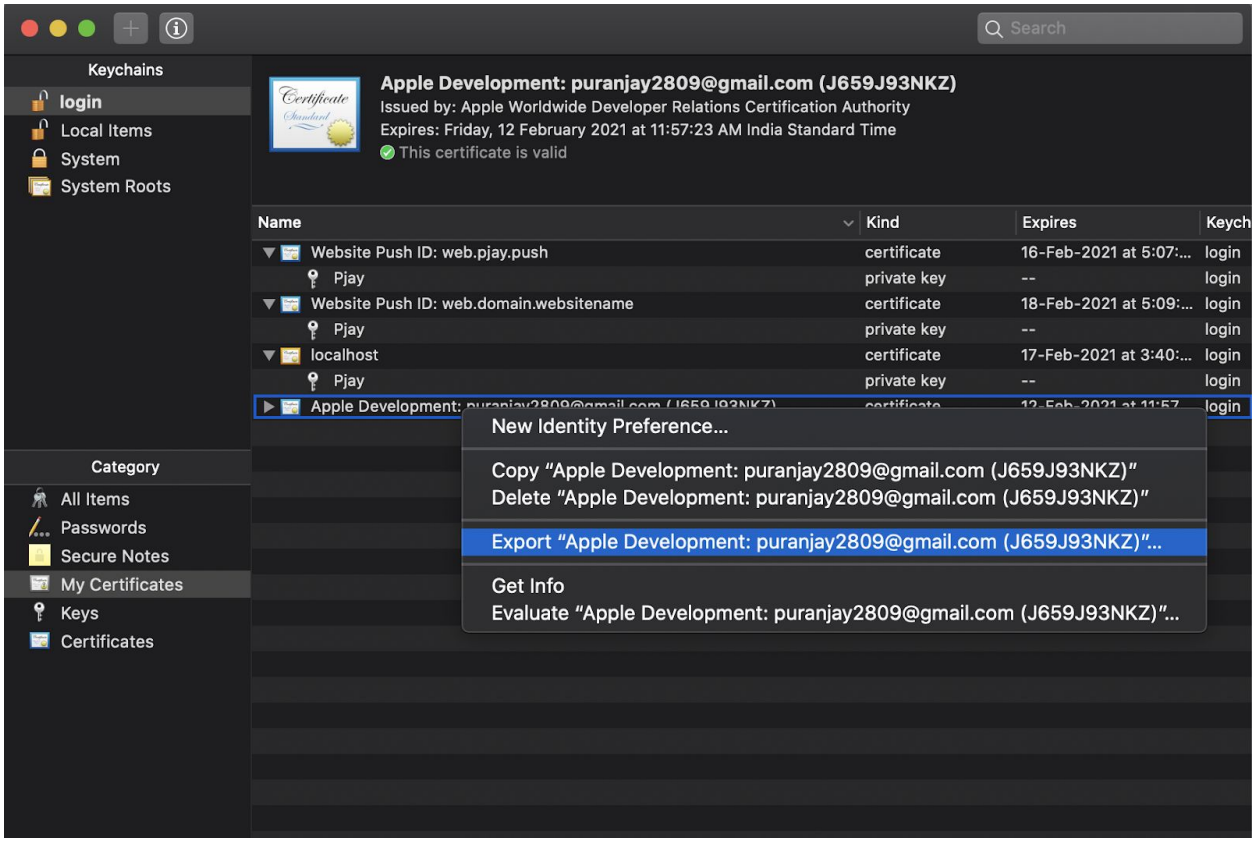
Double click the downloaded certificate and add it to your keychain. Got to your Keychain → My Certificate , select your certificate and private key and export it as **.p12**. Add a password (Don't Forget it, you will need it in a short time).


Search
Keychains
login
Local Items
System
System Roots
Name
Kind
Expires
Keych
Website Push ID: web.pjay.push
certificate
16-Feb-2021 at 5:07:...
login
Pjay
private key
--
login
Website Push ID: web.domain.websitename
certificate
18-Feb-2021 at 5:09:...
login
Pjay
private key
--
login
localhost
certificate
17-Feb-2021 at 3:40:...
login
Pjay
private key
--
login
Apple Development: puranjay2809@gmail.com (J659J93NKZ)
certificate
12-Feb-2021 at 11:57...
login
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates


Enter a password which will be used to protect the exported items:
Password:
Verify:
Password Strength: Weak
Show password
Cancel
OK
Search
Keychains
login
Local Items
System
System Roots
Kind
Expires
certificate
16-Feb-2021 at 5:07:...
private key
--
certificate
18-Feb-2021 at 5:09:...
private key
--
certificate
17-Feb-2021 at 3:40:...
private key
--
Apple Development: puranjay2809@gmail.com (J659J93NKZ)
certificate
12-Feb-2021 at 11:57...
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates

One more certificate named "*Apple Worldwide Developer Relations Certification Authority*" will be needed. Which one can download from https://developer.apple.com/certificationauthority/AppleWWDRCA.cer

Add AWDRCA certificate to the keychain and export it as **.pem** file.
After February 14, 2016, you will need to sign the push package with both the web push certificate and the intermediate certificate.



- Preparing Push Package

When a user is asked for permission to receive push notifications, Safari asks your web server for a package. The package contains data that is used by the notification UI, such as your website name and icon, as well as a cryptographic signature. The signature verifies that your notification hasn't been intercepted by a man-in-the-middle attack and that it is indeed coming from a trusted source.

Below is the folder structure we have to create:

pushpackage/
icon.iconset/
icon_16x16.png
icon_16x16@2x.png
icon_32x32.png

icon_32x32@2x.png
icon_128x128.png
icon_128x128@2x.png
manifest.json
signature
website.json

Website JSON

The website JSON dictionary named website.json contains metadata used by Safari and Notification Center to present UI to the user and to communicate with your web service**.**

Key	Description
websiteName	The website name. This is the heading used in Notification Center.
websitePushID	The Website Push ID, as specified in your developer account.
allowedDomains	An array of websites that are allowed to request permission from the user.
urlFormatString	The URL to go to when the notification is clicked. Use %@ as a placeholder for arguments you fill in when delivering your notification. This URL must use the http or https scheme; otherwise, it is invalid.
authenticationToken	A string that helps you identify the user. It is included in later requests to your web service. This string must be 16 characters or greater.
webServiceURL	The location used to make requests to your web service. The trailing slash should be omitted. This should be https

A sample valid website.json file

```
{
  "websiteName": "Bay Airlines",
  "websitePushID": "web.com.example.domain",
  "allowedDomains": ["http://domain.example.com"],
  "urlFormatString": "http://domain.example.com/%@/?flight=%@",
  "authenticationToken": "19f8d7a6e9fb8a7f6d9330dabe",
  "webServiceURL": "https://example.com/push"
}
```

The Iconset

The iconset is a directory called `icon.iconset` that contains PNG images in varying sizes. The images in your iconset populate the icons displayed to the user in the permission prompt, Notification Center, and the notification itself. Because your icon is static, it is unnecessary to include it in every push notification. Instead, your icons are downloaded once from your server and stored on the user's computer.
The icons should be valid and as per apple guidelines. Use https://tinypng.com/ to compress your icons to the required size.

The Manifest

The manifest is a JSON dictionary named `manifest.json` that contains an entry for each file, where the local file path is the entry's key, and a dictionary object is the entry's value. This dictionary contains the `hashType` and `hashValue`, which is the file's SHA512 checksum.

The Signature

The signature is a PKCS #7 detached signature of the manifest file. Sign the manifest file with the private key associated with your web push certificate that you obtained while registering with Apple.

We will use a PHP file named **createPushPackage.php** to generate the zip file for the above pushpackage folder. The folder to create push package is present in the solution under folder named "safari_push_package_creator"

Step 1:

Copy and paste following files in one folder

- .p12 certificate we created above by registering
- PHP file
- .pem file created using AWDRCA certificate
- Create a tmp folder. Then pushPackage inside it.

Step 2:

Create one pushpackage.raw folder and paste following files in it

- icon.iconset folder and all the icons mentioned above in it
- Website.json as per the guidelines mentioned above

Step 3:

Open the PHP file and modify it as per the following instructions.

- Change $certificate_path to path of .p12 certificate
- Change $certificate_password to password of .p12 certificate
- Replace "<path to .pem>" under create_signature with path to .pem file
- Change $package_dir to path to tmp/pushPackage directory

Step 4:

Once the above steps are done, run the PHP file in the terminal
php createPushPackage.php

If run successfully, under the tmp folder a pushPackage.zip file will be created. You can place this pushPackage.zip in your web service directory which will send the pushPackage.zip to the client when a request for safari push notifications will be requested.

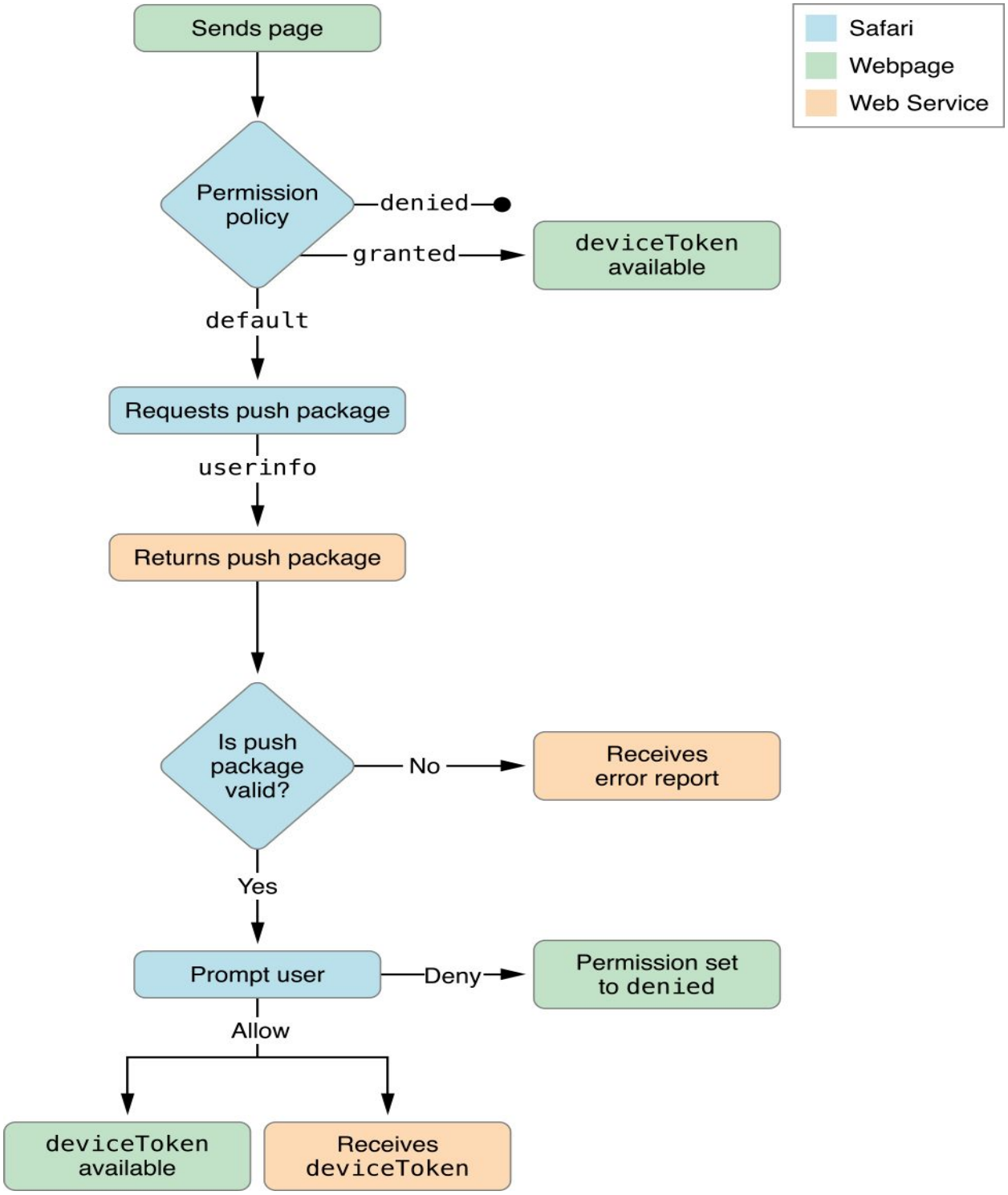
Things to take care of:

1. All the files to be present
2. Use the zipping method used in PHP file attached as order of the files is also important
3. Icons should be valid

4. Use the valid certificates. If it expires, renew it.
5. Use AWDRCA.pem file as mentioned in step 3 as that is mandatory after 2014.

3. Code overview

In this section, we will discuss the client side and web service code which will return the pushPackage.zip file to the client(safari browser). The flow is as per below:



- Client

To request permission to send the user push notifications, call `window.safari.pushNotification.requestPermission()`. Requesting permission is an asynchronous call.
window.safari.pushNotification.requestPermission(url, websitePushID, userInfo, callback);

A description of each argument is as follows:

- `url`—The URL of the web service, which must start with `https`. The web server does not need to be the same domain as the website requesting permission.
- `websitePushID`—The Website Push ID, which must start with the web..
- `userInfo`—An object to pass to the server. Include any data in this object that helps you identify the user requesting permission.
- `callback`—A callback function, which is invoked upon completion.

Sample code:

```
var p = document.getElementById("foo");
p.onclick = function() {
    // Ensure that the user can receive Safari Push Notifications.
    if ('safari' in window && 'pushNotification' in window.safari) {
        var permissionData = window.safari.pushNotification.permission('web.com.example.domain');
        checkRemotePermission(permissionData);
    }
};

var checkRemotePermission = function (permissionData) {
    if (permissionData.permission === 'default') {
        // This is a new web service URL and its validity is unknown.
        window.safari.pushNotification.requestPermission(
            'https://domain.example.com', // The web service URL.
            'web.com.example.domain',     // The Website Push ID.
            {}, // Data that you choose to send to your server to help you identify the user.
            checkRemotePermission         // The callback function.
        );
    }
    else if (permissionData.permission === 'denied') {
        // The user said no.
    }
    else if (permissionData.permission === 'granted') {
        // The web service URL is a valid push provider, and the user said yes.
        // permissionData.deviceToken is now available to use.
    }
```

- Webserver

When a web page requests permission to display push notifications, an HTTP request for your credentials is sent to your web server. Similarly, when a user changes their website push notification settings in Safari or System Preferences, an HTTP request is sent to your web server. You need to configure a RESTful https web service on your server to respond to these requests accordingly. The web service does not need to be hosted on the same server(s) or domain(s) that serve your webpages.

A web service's endpoint URL fragments(this is handled by SafariPushPackageService.java class)

Fragment	Description
webServiceURL	The URL to your web service; this is the same as the url parameter in window.safari.pushNotification.requestPermission(). *Must start with https.*
version	The version of the API. Currently, "v1".
deviceToken	The unique identifier for the user on the device.
websitePushID	The Website Push ID.

1. Downloading Your Website Package
This POST request contains the following information:
The url format is – webServiceURL/version/pushPackages/websitePushID
For us it will be – *https://yourwebsite.com/v1/pushPackages/web.com.yourwebsite*
2. Registering or Updating Device Permission Policy
This POST request contains the following information:
The url format is – webServiceURL/version/devices/deviceToken/registrations/websitePushID
For us it will be –
https://yourwebsite.com/v1/devices/deviceToken/registrations/web.com.yourwebsite
3. Forgetting Device Permission Policy
This DELETE request contains the following information:
The url format is – webServiceURL/version/devices/deviceToken/registrations/websitePushID
For us it will be –
https://yourwebsite.com/v1/devices/deviceToken/registrations/web.com.yourwebsite
4. Logging Errors
If an error occurs, a POST request is sent to the following URL:
The url format is – webServiceURL/version/log
For us it will be — *https://yourwebsite.com/v1/log*

4. Troubleshooting

something goes wrong in downloading your push package or delivery of your push notifications, the logging endpoint on your web service as described in Logging Errors will be contacted with an error message describing the error. Table lists the possible errors and steps you can take to fix them.

Error Message	Resolution
AuthenticationToken must be at least 16 characters.	The authenticationToken key in your website.json file must be 16 characters or greater. See The Website JSON Dictionary.
Downloading push notification package failed.	The push package could not be retrieved from the location specified in Downloading Your Website Package.
Extracting push notification package failed.	Make sure that your push package is zipped correctly. See Building the Push Package.
Missing file in push notification package.	Make sure that your push package contains all of the files specified in Building the Push Package.
Missing image in push notification package.	Make sure that your push package contains all of the files specified in The Iconset.
Missing key in website.json.	Make sure that your website.json file has all of the keys listed in The Website JSON Dictionary.
Serialization of JSON in website.json failed.	The website.json file in your push package is not valid JSON. See The Website JSON Dictionary.
Signature verification of push package failed.	The manifest was not signed correctly or was signed using an invalid certificate. See The Signature.
Unable to create notification bundle for push notification package.	The extracted push package could not be saved to the user's disk.

Unable to generate ICNS file for push notification package.	Your iconset may have malformed PNGs. See The Iconset.
Unable to parse webServiceURL.	Make sure that the value for webServiceURL in your website.json file is a valid URL. See The Website JSON Dictionary.
Unable to save push notification package.	The push package could not be saved to the user's disk.
urlFormatString must have http or https scheme.	Make sure that the value for urlFormatString in your website.json file starts with http or https. See The Website JSON Dictionary.
Verifying hashes in manifest.json failed.	The SHA512 checksums specified in your manifest.json file do not compute to their actual values. See The Manifest.
Web Service API URL must be https.	Make sure that the URL at your push package endpoint starts with https. See Downloading Your Website Package.
webServiceURL must be equal to URL in call to requestPermission.	Cross-check that the web service URL in your JavaScript call matches the web service URL in your website.json file. See Requesting Permission.
websitePushID must be equal to identifier in call to requestPermission.	Cross-check that the identifier in your JavaScript call matches the identifier in your website.json file. See Requesting Permission.
x cannot be used as a format string for URLs.	The URL created by inserting the notification payload's url-args values into the placeholders specified in the push package's urlFormatString is not valid, or there are a different number of values than placeholders. See Pushing Notifications.

5. Errors while sending web push

Invalid Authentication,
BadRequest(400),
DeviceTokenInactiveForTopic(410),
ServerUnavailable(503)

6. Limitations

There are certain limitations with Safari web push as compared to Chrome or Firefox. Below are the technical limitations listed for the same:-

a. Deep link URL is mandatory or else the notifications will not be rendered.
b. Custom icons won’t be rendered in the notifications as Safari uses icon present in the Safari push package
c. Call to action buttons are not supported
d. Image is not supported in the notification
e. Sent and Viewed events are not supported as there is no handler unlike Service Worker in Chrome and Firefox to capture the click and view events