Sending Apple Push Notifications ...

Random Thoughts and Musings of a Mobile Application Developer

Sending Apple Push Notifications (APN) in C - Updated (Using CA Cert) Thu, Dec 9. 2010

Update 22.01.2012: Thanks to asmera I discovered, that Apple seems to be stricter with the JSON you send and that since a few days the Apple Servers do not accept the output from the example given here. I changed the code in here as well as in the downloadable example to make it work again.

Today I will show you a quick implementation of an Apple Push Notification Server provider in C. (Now updated with the new Entrust CA cert and instructions how to use it. For information on how to trust it go **here**) Some parts of this (like creating the needed certificates) are taken from **this** very good tutorial about building the same thing in php. For a good intro on the topic and how APNs works internally you should first read **this**.

You're done? Good, then let's get started:

First you need to create the Push Certificates and enable your application to get notified:

1. Log in to the iPhone Developer Connection Portal and click App IDs Ensure you have created an App ID without a wildcard. Wildcard IDs cannot use the push notification service. For example, our iPhone application ID looks something like AB123346CD.com.serverdensity.iphone

2. Click Configure next to your App ID and then click the button to generate a Push Notification certificate. A wizard will appear guiding you through the steps to generate a signing authority and then upload it to the portal, then download the newly generated certificate. This step is also covered in the Apple documentation.

3. Import your aps_developer_identity.cer into your Keychain by double clicking the .cer file. Launch Keychain Assistant from your local Mac and from the login keychain, filter by the Certificates category. You will see an expandable option called "Apple Development Push Services"

4. Expand this option then right click on "Apple Development Push Services" > Export "Apple Development Push Services ID123". Save this as apns-dev-cert.p12 file somewhere you can access it.

5. Do the same again for the "Private Key" that was revealed when you expanded "Apple Development Push Services" ensuring you save it as apns-dev-key.p12 file.

6. These files now need to be converted to the PEM format by executing this command from the terminal:

openssl pkcs12 -clcerts -nokeys -out apns-dev-cert.pem -in apns-dev-cert.p12
openssl pkcs12 -nocerts -out apns-dev-key-enc.pem -in apns-dev-key.p12

Then to remove the passphrase execute:

openssl rsa -in apns-dev-key-enc.pem -out apns-dev-key.pem

Once we are finished, this little example program will help us testing if we did everything right.

```
1.
2.  #include <stdio.h>
3.  #include <string.h>
4.  #include <errno.h>
5.
6.  #include "Helper/RemoteNotification.h"
7.
8.  /* MAIN Function */
9.  int main(int argc, char *argv[])
10. {
11.     int     err;
12.
13.     /* Phone specific Payload message as well as hex formated device token */
```

```
14.     const char      *deviceTokenHex = NULL;
15.     if(argc == 1)
16.     {
17.         deviceTokenHex = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
18.     }
19.     else
20.     {
21.         deviceTokenHex = argv[1];
22.     }
23.
24.     if(strlen(deviceTokenHex) < 64 || strlen(deviceTokenHex) > 70)
25.     {
26.         printf("Device Token is to short or to long. Length without spaces should be 64 chars...\n");
27.         exit(1);
28.     }
29.
30.     Payload *payload = (Payload*)malloc(sizeof(Payload));
31.     init_payload(payload);
32.
33.     // This is the message the user gets once the Notification arrives
34.     payload->message = "Message to print out";
35.
36.     // This is the red numbered badge that appears over the Icon
37.     payload->badgeNumber = 1;
38.
39.     // This is the Caption of the Action key on the Dialog that appears
40.     payload->actionKeyCaption = "Caption of the second Button";
41.
42.     // These are two dictionary key-value pairs with user-content
43.     payload->dictKey[0] = "Key1";
44.     payload->dictValue[0] = "Value1";
45.
46.     payload->dictKey[1] = "Key2";
47.     payload->dictValue[1] = "Value2";
48.
49.     /* Send the payload to the phone */
50.     printf("Sending APN to Device with UDID: %s\n", deviceTokenHex);
51.     send_remote_notification(deviceTokenHex, payload);
52.
53.     return 0;
54. }
55.
```

send_remote_notification does everything we need. It first prepares the payload, which is encapsulated into a struct:

```
1.
2. typedef struct {
3.     /* The Message that is displayed to the user */
4.     char *message;
5.
6.     /* The name of the Sound which will be played back */
7.     char *soundName;
8.
9.     /* The Number which is plastered over the icon, 0 disables it */
10.    int badgeNumber;
11.
12.    /* The Caption of the Action Key the user needs to press to launch the Application */
13.    char *actionKeyCaption;
14.
15.    /* Custom Message Dictionary, which is accessible from the Application */
16.    char* dictKey[5];
17.    char* dictValue[5];
18. } Payload;
19.
20. #define DEVICE_BINARY_SIZE  32
21. #define MAXPAYLOAD_SIZE     256
22.
```

From the given Information JSON is generated (JSON is described **here**) and then sent. It is implemented as follows:

```
int send_remote_notification(const char *deviceTokenHex, Payload *payload)
{
        char messageBuff[MAXPAYLOAD_SIZE];
        char tmpBuff[MAXPAYLOAD_SIZE];
        char badgenumBuff[3];

        strcpy(messageBuff, "{\"aps\":{");

        if(payload->message != NULL)
        {
            strcat(messageBuff, "\"alert\":");
            if(payload->actionKeyCaption != NULL)
            {
                sprintf(tmpBuff, "{\"body\":\"%s\",\"action-loc-key\":\"%s\"},", payload->message, payload->actionKeyCaption);
                strcat(messageBuff, tmpBuff);
            }
            else
            {
                sprintf(tmpBuff, "{\"%s\"},", payload->message);
                strcat(messageBuff, tmpBuff);
            }
        }

        if(payload->badgeNumber > 99 || payload->badgeNumber < 0)
            payload->badgeNumber = 1;

        sprintf(badgenumBuff, "%d", payload->badgeNumber);
        strcat(messageBuff, "\"badge\":");
        strcat(messageBuff, badgenumBuff);

        strcat(messageBuff, ",\"sound\":\"");
        strcat(messageBuff, payload->soundName == NULL ? "default" : payload->soundName);
        strcat(messageBuff, "\"},");

        int i = 0;
        while(payload->dictKey[i] != NULL && i < 5)
        {
            sprintf(tmpBuff, "\"%s\":\"%s\"", payload->dictKey[i], payload->dictValue[i]);
            strcat(messageBuff, tmpBuff);
            if(i < 4 && payload->dictKey[i + 1] != NULL)
            {
                strcat(messageBuff, ",");
            }
            i++;
        }

        strcat(messageBuff, "}");
        printf("Sending %s\n", messageBuff);

        send_payload(deviceTokenHex, messageBuff, strlen(messageBuff));
}
```

It allows up to 5 custom key-value pairs in a custom dictionary. After the method is done formatting, the sending process is initiated by calling the *send_payload* function. The payload can be sent either through the development push server or the production push server, depending on the used certificate. For this purpose a define is used, which selects the right type depending on the build.

```
    #define CA_CERT_PATH    "/path/to/the/ca/cert"

#if defined(IS_DEVELOPMENT_VERSION)
    /* Development Certificates */
    #define RSA_CLIENT_CERT     "Certs/apns-dev-cert.pem"
    #define RSA_CLIENT_KEY      "Certs apns-dev-key.pem"

    /* Development Connection Infos */
    #define APPLE_HOST          "gateway.sandbox.push.apple.com"
    #define APPLE_PORT          2195

    #define APPLE_FEEDBACK_HOST "feedback.sandbox.push.apple.com"
```

```
    #define APPLE_FEEDBACK_PORT 2196
#else
    /* Distribution Certificates */
    #define RSA_CLIENT_CERT     "Certs/apns-dis-cert.pem"
    #define RSA_CLIENT_KEY      "Certs/apns-dis-key.pem"

    /* Release Connection Infos */
    #define APPLE_HOST          "gateway.push.apple.com"
    #define APPLE_PORT          2195

    #define APPLE_FEEDBACK_HOST "feedback.push.apple.com"
    #define APPLE_FEEDBACK_PORT 2196
#endif


int send_payload(const char *deviceTokenHex, const char *payloadBuff, size_t payloadLength)
{
    int rtn = 0;

    SSL_Connection *sslcon = ssl_connect(APPLE_HOST, APPLE_PORT, RSA_CLIENT_CERT, RSA_CLIENT_KEY, CA_CERT_PATH);
    if(sslcon == NULL)
    {
        printf("Could not allocate memory for SSL Connection");
        exit(1);
    }

    if (sslcon && deviceTokenHex && payloadBuff && payloadLength)
    {
        uint8_t command = 0; /* command number */
        char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint16_t) + DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];

        /* message format is, |COMMAND|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */
        char *binaryMessagePt = binaryMessageBuff;
        uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);
        uint16_t networkOrderPayloadLength = htons(payloadLength);

        /* command */
        *binaryMessagePt++ = command;

        /* token length network order */
        memcpy(binaryMessagePt, &networkOrderTokenLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

        /* Convert the Device Token */
        int i = 0;
        int j = 0;
        int tmpi;
        char tmp[3];
        char deviceTokenBinary[DEVICE_BINARY_SIZE];
        while(i < strlen(deviceTokenHex))
        {
            if(deviceTokenHex[i] == ' ')
            {
                i++;
            }
            else
            {
                tmp[0] = deviceTokenHex[i];
                tmp[1] = deviceTokenHex[i + 1];
                tmp[2] = '\0';

                sscanf(tmp, "%x", &tmpi);
                deviceTokenBinary[j] = tmpi;

                i += 2;
                j++;
            }
        }

        /* device token */
        memcpy(binaryMessagePt, deviceTokenBinary, DEVICE_BINARY_SIZE);
```

```
        binaryMessagePt += DEVICE_BINARY_SIZE;

        /* payload length network order */
        memcpy(binaryMessagePt, &networkOrderPayloadLength, sizeof(uint16_t));
        binaryMessagePt += sizeof(uint16_t);

        /* payload */
        memcpy(binaryMessagePt, payloadBuff, payloadLength);
        binaryMessagePt += payloadLength;
        if (SSL_write(sslcon->ssl, binaryMessageBuff, (binaryMessagePt - binaryMessageBuff)) > 0)
            rtn = 1;
    }

    ssl_disconnect(sslcon);

    return rtn;
}
```

We're nearly done with the server part, the last (and hardest) puzzle piece is the SSL connection, which is wrapped in a neat structure:

```
typedef struct {
    /* SSL Vars */
    SSL_CTX         *ctx;
    SSL             *ssl;
    SSL_METHOD      *meth;
    X509            *server_cert;
    EVP_PKEY        *pkey;

    /* Socket Communications */
    struct sockaddr_in  server_addr;
    struct hostent      *host_info;
    int                 sock;
} SSL_Connection;
```

So let's get right to using it:

```
SSL_Connection *ssl_connect(const char *host, int port, const char *certfile, const char *keyfile, const char* capath)
{
    int err;

    SSL_Connection *sslcon = NULL;
    sslcon = (SSL_Connection *)malloc(sizeof(SSL_Connection));
    if(sslcon == NULL)
    {
        printf("Could not allocate memory for SSL Connection");
        exit(1);
    }

    /* Load encryption & hashing algorithms for the SSL program */
    SSL_library_init();

    /* Load the error strings for SSL & CRYPTO APIs */
    SSL_load_error_strings();

    /* Create an SSL_METHOD structure (choose an SSL/TLS protocol version) */
    sslcon->meth = SSLv3_method();

    /* Create an SSL_CTX structure */
    sslcon->ctx = SSL_CTX_new(sslcon->meth);
    if(!sslcon->ctx)
    {
        printf("Could not get SSL Context\n");
        exit(1);
    }

    /* Load the CA from the Path */
    if(SSL_CTX_load_verify_locations(sslcon->ctx, NULL, capath) <= 0)
```

```
33.     {
34.         /* Handle failed load here */
35.         printf("Failed to set CA location...\n");
36.         ERR_print_errors_fp(stderr);
37.         exit(1);
38.     }
39.
40.     /* Load the client certificate into the SSL_CTX structure */
41.     if (SSL_CTX_use_certificate_file(sslcon->ctx, certfile, SSL_FILETYPE_PEM) <= 0) {
42.         printf("Cannot use Certificate File\n");
43.         ERR_print_errors_fp(stderr);
44.         exit(1);
45.     }
46.
47.     /* Load the private-key corresponding to the client certificate */
48.     if (SSL_CTX_use_PrivateKey_file(sslcon->ctx, keyfile, SSL_FILETYPE_PEM) <= 0) {
49.         printf("Cannot use Private Key\n");
50.         ERR_print_errors_fp(stderr);
51.         exit(1);
52.     }
53.
54.     /* Check if the client certificate and private-key matches */
55.     if (!SSL_CTX_check_private_key(sslcon->ctx)) {
56.         printf("Private key does not match the certificate public key\n");
57.         exit(1);
58.     }
59.
60.     /* Set up a TCP socket */
61.     sslcon->sock = socket (PF_INET, SOCK_STREAM, IPPROTO_TCP);
62.     if(sslcon->sock == -1)
63.     {
64.         printf("Could not get Socket\n");
65.         exit(1);
66.     }
67.
68.     memset (&sslcon->server_addr, '\0', sizeof(sslcon->server_addr));
69.     sslcon->server_addr.sin_family      = AF_INET;
70.     sslcon->server_addr.sin_port        = htons(port);       /* Server Port number */
71.     sslcon->host_info = gethostbyname(host);
72.     if(sslcon->host_info)
73.     {
74.         /* Take the first IP */
75.         struct in_addr *address = (struct in_addr*) sslcon->host_info->h_addr_list[0];
76.         sslcon->server_addr.sin_addr.s_addr = inet_addr(inet_ntoa(*address)); /* Server IP */
77.     }
78.     else
79.     {
80.         printf("Could not resolve hostname %s\n", host);
81.         return NULL;
82.     }
83.
84.     /* Establish a TCP/IP connection to the SSL client */
85.     err = connect(sslcon->sock, (struct sockaddr*) &sslcon->server_addr, sizeof(sslcon->server_addr));
86.     if(err == -1)
87.     {
88.         printf("Could not connect\n");
89.         exit(1);
90.     }
91.
92.     /* An SSL structure is created */
93.     sslcon->ssl = SSL_new(sslcon->ctx);
94.     if(!sslcon->ssl)
95.     {
96.         printf("Could not get SSL Socket\n");
97.         exit(1);
98.     }
99.
100.    /* Assign the socket into the SSL structure (SSL and socket without BIO) */
101.    SSL_set_fd(sslcon->ssl, sslcon->sock);
102.
103.    /* Perform SSL Handshake on the SSL client */
104.    err = SSL_connect(sslcon->ssl);
```

```
105.     if(err <= 0)
106.     {
107.         printf("Could not connect to SSL Server\n");
108.         exit(1);
109.     }
110.
111.     return sslcon;
112. }
113.
114. void ssl_disconnect(SSL_Connection *sslcon)
115. {
116.     int err;
117.
118.     if(sslcon == NULL)
119.     {
120.         return;
121.     }
122.
123.     /* Shutdown the client side of the SSL connection */
124.     err = SSL_shutdown(sslcon->ssl);
125.     if(err == -1)
126.     {
127.         printf("Could not shutdown SSL\n");
128.         exit(1);
129.     }
130.
131.     /* Terminate communication on a socket */
132.     err = close(sslcon->sock);
133.     if(err == -1)
134.     {
135.         printf("Could not close socket\n");
136.         exit(1);
137.     }
138.
139.     /* Free the SSL structure */
140.     SSL_free(sslcon->ssl);
141.
142.     /* Free the SSL_CTX structure */
143.     SSL_CTX_free(sslcon->ctx);
144.
145.     /* Free the sslcon */
146.     if(sslcon != NULL)
147.     {
148.         free(sslcon);
149.         sslcon = NULL;
150.     }
151. }
152.
```