Legacy Notification Format

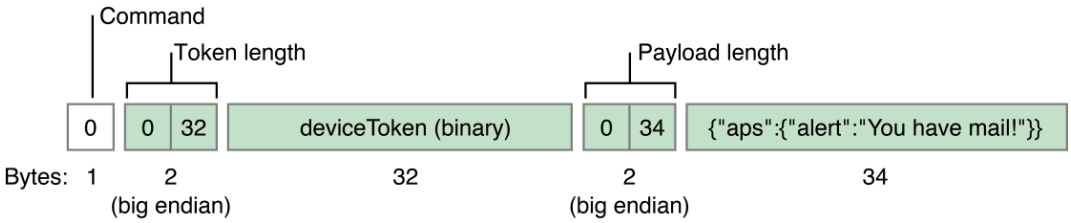
New development should use the modern format to connect to APNs, as described in APNs Provider API.

These formats do not include a priority; a priority of `10` is assumed.

Legacy Notification Format

Figure B-1 shows this format.

Figure B-1 Legacy notification format



The first byte in the legacy format is a command value of 0 (zero). The other fields are the same as the enhanced format. Listing B-1 gives an example of a function that sends a remote notification to APNs over the binary interface using the legacy notification format. The example assumes prior SSL connection to `gateway.push.apple.com` (or `gateway.sandbox.push.apple.com`) and peer-exchange authentication.

Listing B-1 Sending a notification in the legacy format via the binary interface

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff, size_t
payloadLength)
{
    bool rtn = false;
    if (sslPtr && deviceTokenBinary && 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);

        /* 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(sslPtr, binaryMessageBuff, (binaryMessagePt - binaryMessageBuff)) >
0)
            rtn = true;
    }
    return rtn;
}
```

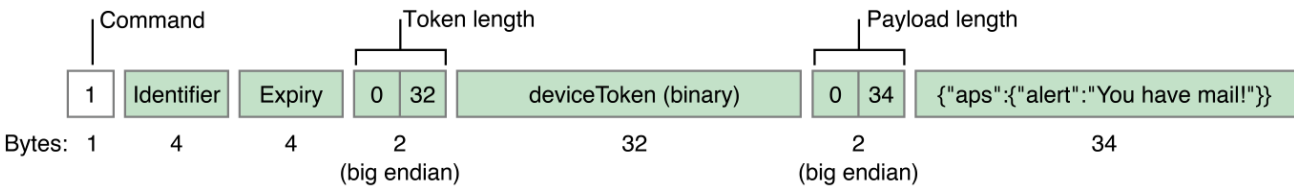
Enhanced Notification Format

The enhanced format has several improvements over the legacy format:

- **Error response**. With the legacy format, if you send a notification packet that is malformed in some way—for example, the payload exceeds the stipulated limit—APNs responds by severing the connection. It gives no indication why it rejected the notification. The enhanced format lets a provider tag a notification with an arbitrary identifier. If there is an error, APNs returns a packet that associates an error code with the identifier. This response enables the provider to locate and correct the malformed notification.
- **Notification expiration**. APNs has a store-and-forward feature that keeps the most recent notification sent to an app on a device. If the device is offline at time of delivery, APNs delivers the notification when the device next comes online. With the legacy format, the notification is delivered regardless of the pertinence of the notification. In other words, the notification can become "stale" over time. The enhanced format includes an expiry value that indicates the period of validity for a notification. APNs discards a notification in store-and-forward when this period expires.

Figure B-2 depicts the format for notification packets.

Figure B-2 Enhanced notification format



The first byte in the notification format is a command value of 1. The remaining fields are as follows:

- **Identifier**—An arbitrary value that identifies this notification. This same identifier is returned in a error-response packet if APNs cannot interpret a notification.
- **Expiry**—A fixed UNIX epoch date expressed in seconds (UTC) that identifies when the notification is no longer valid and can be discarded. The expiry value uses network byte order (big endian). If the expiry value is non-zero, APNs tries to deliver the notification at least once. Specify zero to request that APNs not store the notification at all.
- **Token length**—The length of the device token in network order (that is, big endian)
- **Device token**—The device token in binary form.
- **Payload length**—The length of the payload in network order (that is, big endian). The payload must not exceed 256 bytes and must *not* be null-terminated.
- **Payload**—The notification payload.

Listing B-2 composes a remote notification in the enhanced format before sending it to APNs. It assumes prior SSL connection to `gateway.push.apple.com` (or `gateway.sandbox.push.apple.com`) and peer-exchange authentication.

Listing B-2 Sending a notification in the enhanced format via the binary interface

```
static bool sendPayload(SSL *sslPtr, char *deviceTokenBinary, char *payloadBuff, size_t
payloadLength)
{
  bool rtn = false;
  if (sslPtr && deviceTokenBinary && payloadBuff && payloadLength)
  {
      uint8_t command = 1; /* command number */
      char binaryMessageBuff[sizeof(uint8_t) + sizeof(uint32_t) + sizeof(uint32_t) +
sizeof(uint16_t) +
          DEVICE_BINARY_SIZE + sizeof(uint16_t) + MAXPAYLOAD_SIZE];
      /* message format is, |COMMAND|ID|EXPIRY|TOKENLEN|TOKEN|PAYLOADLEN|PAYLOAD| */
      char *binaryMessagePt = binaryMessageBuff;
      uint32_t whicheverOrderIWantToGetBackInAErrorResponse_ID = 1234;
      uint32_t networkOrderExpiryEpochUTC = htonl(time(NULL)+86400); // expire message if
not delivered in 1 day
      uint16_t networkOrderTokenLength = htons(DEVICE_BINARY_SIZE);
      uint16_t networkOrderPayloadLength = htons(payloadLength);

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

     /* provider preference ordered ID */
     memcpy(binaryMessagePt, &whicheverOrderIWantToGetBackInAErrorResponse_ID,
sizeof(uint32_t));
     binaryMessagePt += sizeof(uint32_t);

     /* expiry date network order */
     memcpy(binaryMessagePt, &networkOrderExpiryEpochUTC, sizeof(uint32_t));
     binaryMessagePt += sizeof(uint32_t);

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

      /* 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(sslPtr, binaryMessageBuff, (binaryMessagePt - binaryMessageBuff)) >
```

```
0)
          rtn = true;
  }
  return rtn;
}
```