delivering this token to its content provider.

APNs servers also have the necessary certificates, CA certificates, and cryptographic keys (private and public) for validating connections and the identities of providers and devices.

The Notification Payload

Each push notification carries with it a payload. The payload specifies how users are to be alerted to the data waiting to be downloaded to the client application. The maximum size allowed for a notification payload is 256 bytes; Apple Push Notification Service refuses any notification that exceeds this limit. Remember that delivery of notifications is “best effort” and is not guaranteed.

For each notification, providers must compose a JSON dictionary object that strictly adheres to RFC 4627. This dictionary must contain another dictionary identified by the key `aps`. The `aps` dictionary contains one or more properties that specify the following actions:

- An alert message to display to the user
- A number to badge the application icon with
- A sound to play

Note: Although you can combine an alert message, icon badging, and a sound in a single notification, you should consider the human-interface implications with push notifications. For example, a user might find frequent alert messages with accompanying sound more annoying than useful, especially when the data to be downloaded is not critical.

If the target application isn’t running when the notification arrives, the alert message, sound, or badge value is played or shown. If the application is running, iOS delivers it to the application delegate as an `NSDictionary` object. The dictionary contains the corresponding Cocoa property-list objects (plus `NSNull`).

Providers can specify custom payload values outside the Apple-reserved `aps` namespace. Custom values must use the JSON structured and primitive types: dictionary (object), array, string, number, and Boolean. You should not include customer information as custom payload data. Instead, use it for such purposes as setting context (for the user interface) or internal metrics. For example, a custom payload value might be a conversation identifier for use by an instant-message client application or a timestamp identifying when the provider sent the notification. Any action associated with an alert message should not be destructive—for example, deleting data on the device.

Important: Because delivery is not guaranteed, you should not depend on the remote-notifications facility for delivering critical data to an application via the payload. And never include sensitive data in the payload. You should use it only to *notify* the user that new data is available.

Table 3-1 lists the keys and expected values of the `aps` payload.

Table 3-1 Keys and values of the `aps` dictionary

Key	Value type	Comment

`alert`	string or dictionary	If this property is included, iOS displays a standard alert. You may specify a string as the value of `alert` or a dictionary as its value. If you specify a string, it becomes the message text of an alert with two buttons: Close and View. If the user taps View, the application is launched. Alternatively, you can specify a dictionary as the value of `alert`. See Table 3-2 for descriptions of the keys of this dictionary.
`badge`	number	The number to display as the badge of the application icon. If this property is absent, any badge number currently shown is removed.
`sound`	string	The name of a sound file in the application bundle. The sound in this file is played as an alert. If the sound file doesn't exist or `default` is specified as the value, the default alert sound is played. The audio must be in one of the audio data formats that are compatible with system sounds; see "Preparing Custom Alert Sounds" for details.

Table 3-2 Child properties of the `alert` property

Key	Value type	Comment
`body`	string	The text of the alert message.
`action-loc-key`	string or `null`	If a string is specified, displays an alert with two buttons, whose behavior is described in Table 3-1. However, iOS uses the string as a key to get a localized string in the current localization to use for the right button's title instead of "View". If the value is `null`, the system displays an alert with a single OK button that simply dismisses the alert when tapped. See "Localized Formatted Strings" for more information.
`loc-key`	string	A key to an alert-message string in a `Localizable.strings` file for the current localization (which is set by the user's language preference). The key string can be formatted with `%@` and `%`*n*`$@` specifiers to take the variables specified in `loc-args`. See "Localized Formatted Strings" for more information.
`loc-args`	array of strings	Variable string values to appear in place of the format specifiers in `loc-key`. See "Localized Formatted Strings" for more information.
`launch-image`	string	The filename of an image file in the application bundle; it may include the extension or omit it. The image is used as the launch image when users tap the action button or move the action slider. If this property is not specified, the system either uses the previous snapshot,uses the image identified by the `UILaunchImageFile` key in the application's `Info.plist` file, or falls back to `Default.png`. This property was added in iOS 4.0.

Note: If you want the iPhone, iPad, or iPod touch device to display the message text as-is in an alert that has both the Close and View buttons, then specify a string as the direct value of alert. *Don't* specify a dictionary as the value of `alert` if the dictionary only has the `body` property.

Localized Formatted Strings

You can display localized alert messages in two ways. The server originating the notification can localize the text; to do this, it must discover the current language preference selected for the device (see "Passing the Provider the Current Language Preference (Remote Notifications)"). Or the client application can store in its bundle the alert-message strings translated for each localization it supports. The provider specifies the `loc-key` and `loc-args` properties in the `aps` dictionary of the notification payload. When the device receives the notification (assuming the application isn't running), it uses these `aps`-dictionary properties to find and format the string localized for the current language, which it then displays to the user.

Here's how that second option works in a little more detail.

An iOS application can internationalize resources such as images, sounds, and text for each language that it supports, Internationalization collects the resources and puts them in a subdirectory of the bundle with a two-part name: a language code and an extension of `.lproj` (for example, `fr.lproj`). Localized strings that are programmatically displayed are put in a file called `Localizable.strings`. Each entry in this file has a key and a localized string value; the string can have format specifiers for the substitution of variable values. When an application asks for a particular resource—say a localized string—it gets the resource that is localized for the language currently selected by the user. For example, if the preferred language is French, the corresponding string value for an alert message would be fetched from `Localizable.strings` in the `fr.lproj` directory in the application bundle. (iOS makes this request through the `NSLocalizedString` macro.)

Note: This general pattern is also followed when the value of the `action-loc-key` property is a string. This string is a key into the `Localizable.strings` in the localization directory for the currently selected language. iOS uses this key to get the title of the button on the right side of an alert message (the "action" button).

To make this clearer, let's consider an example. The provider specifies the following dictionary as the value of the alert property:

```
"alert" : { "loc-key" : "GAME_PLAY_REQUEST_FORMAT", "loc-args" : [ "Jenna", "Frank"]
},
```

When the device receives the notification, it uses `"GAME_PLAY_REQUEST_FORMAT"` as a key to look up the associated string value in the `Localizable.strings` file in the `.lproj` directory for the current language. Assuming the current localization has an `Localizable.strings` entry such as this:

```
"GAME_PLAY_REQUEST_FORMAT" = "%@ and %@ have invited you to play Monopoly";
```

the device displays an alert with the message "Jenna and Frank have invited you to play Monopoly".

In addition to the format specifier `%@`, you can `%`*n*`$@` format specifiers for positional substitution of string variables. The *n* is the index (starting with 1) of the array value in `loc-args` to substitute. (There's also the `%%` specifier for expressing a percentage sign (%).) So if the entry in

`Localizable.strings` is this:

```
"GAME_PLAY_REQUEST_FORMAT" = "%2$@ and %1$@ have invited you to play Monopoly";
```

the device displays an alert with the message "Frank and Jenna have invited you to play Monopoly".

For a full example of a notification payload that uses the `loc-key` and `loc-arg` properties, see the last example of “Examples of JSON Payloads.” To learn more about internationalization in iOS, see ““Build-Time Configuration Details”” in *iOS Application Programming Guide*; for general information about internationalization, see *Internationalization Programming Topics*. String formatting is discussed in “Formatting String Objects” in *String Programming Guide*.

Note: You should use the `loc-key` and `loc-args` properties—and the `alert` dictionary in general—only if you absolutely need to. The values of these properties, especially if they are long strings, might use up more bandwidth than is good for performance. Many if not most applications may not need these properties because their message strings are originated by users and thus are implicitly "localized."

Examples of JSON Payloads

The following examples of the payload portion of notifications illustrate the practical use of the properties listed in Table 3-1. Properties with “acme” in the key name are examples of custom payload data. The examples include whitespace and newline characters for readability; for better performance, providers should omit whitespace and newline characters.

Example 1: The following payload has an `aps` dictionary with a simple, recommended form for alert messages with the default alert buttons (Close and View). It uses a string as the value of `alert` rather than a dictionary. This payload also has a custom array property.

```
{
    "aps" : { "alert" : "Message received from Bob" },
    "acme2" : [ "bang",  "whiz" ]
}
```

Example 2. The payload in the example uses an `aps` dictionary to request that the device display an alert message with an Close button on the left and a localized title for the "action" button on the right side of the alert. In this case, “PLAY” is used as a key into the `Localizable.strings` file for the currently selected language to get the localized equivalent of “Play”. The `aps` dictionary also requests that the application icon be badged with 5.

```
{     "aps" : {         "alert" : { "body" : "Bob wants to play poker", "action-loc-
key" : "PLAY" },          "badge" : 5,     },     "acme1" : "bar",     "acme2" : [
"bang",  "whiz" ] }
```

Example 3. The payload in this example specifies that device should display an alert message with both Close and View buttons. It also request that the application icon be badged with 9 and that a bundled alert sound be played when the notification is delivered.

```
{
    "aps" : {
        "alert" : "You got your emails.",
        "badge" : 9,
        "sound" : "bingbong.aiff"
    },
    "acme1" : "bar",
    "acme2" : 42
}
```

Example 4. The interesting thing about the payload in this example is that it uses the `loc-key` and `loc-args` child properties of the `alert` dictionary to fetch a formatted localized string from the application's bundle and substitute the variable string values (`loc-args`) in the appropriate places. It also specifies a custom sound and include a custom property.

```
{
    "aps" : {
        "alert" : { "loc-key" : "GAME_PLAY_REQUEST_FORMAT", "loc-args" : [ "Jenna",
"Frank"] },
        "sound" : "chime",
    },
    "acme" : "foo"
}
```

Example 5. The following example shows an empty `aps` dictionary; because the `badge` property is missing, any current badge number shown on the application icon is removed. The `acme2` custom property is an array of two integers.

```
{
    "aps" : {
    },
    "acme2" : [ 5,  8 ]
}
```

Remember, for better performance, you should strip all whitespace and newline characters from the payload before including it in the notification.

Did this document help you? Yes It's good, but... Not helpful...