Publication Subscription Programming Guide

Internet & Web > Web Content

2007-05-11

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and Listings

Introduction

Introduced in Mac OS X v10.5, Publication Subscription is a technology that offers developers a way to subscribe to web feeds from their applications. Web feeds are documents that contains frequently updated information. You can use Publication Subscription to allow your applications to subscribe to podcasts, photocasts, and any other feed-based document. Plus, Publication Subscription handles all the feed downloads and updates automatically. This document explains how feeds work and how to use them in your application.

Before reading this document, you should have some experience with Objective-C and be familiar with XML.

Organization of This Document

This book contains the following chapters:

- "Understanding Feeds" (page 9) describes what feeds are and how they work.
- "Publication Subscription Overview" (page 13) explains the architecture of Publication Subscription.
- "Subscribing to a Feed" (page 17) explains how to use Publication Subscription to subscribe to a feed.
- "Viewing and Retrieving Content" (page 21) describes how to read the data in a feed.

See Also

For an in-depth description of the Publication Subscription framework, read:

- *Publication Subscription Framework Reference*

For more information about some of the technology areas used by Publication Subscription, refer to:

- *Core Foundation Design Concepts* to learn more about the Cocoa design patterns
- *Tree-Based XML Programming Guide for Cocoa* to learn more about XML and how to use it in Cocoa

There are also a number of good websites about the different feed standards supported by Publication Subscription:

- For more information about the RSS 0.9, RSS 1.0 and RSS 2.0 formats, read http://en.wikipedia.org/wiki/Really_Simple_Syndication.
- For more information about the Atom Syndication Format, read http://en.wikipedia.org/wiki/Atom_%28standard%29.

Introduction

See Also

Understanding Feeds

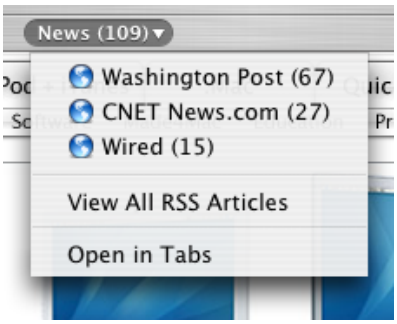
Before you use Publication Subscription, it is important to understand what feeds are and how they work. This chapter explains the data structure of a feed, how a feed is created, the different types of feeds, and how to use different feed namespaces.

What Is a Feed?

A **feed** is an XML document that contains frequently updated information. A feed provides information or data independent of presentation. Thus, an XML document can be parsed by an application to retrieve information without the additional style of an entire webpage. Additionally, the parsing application can determine what information is new and mark it as such.

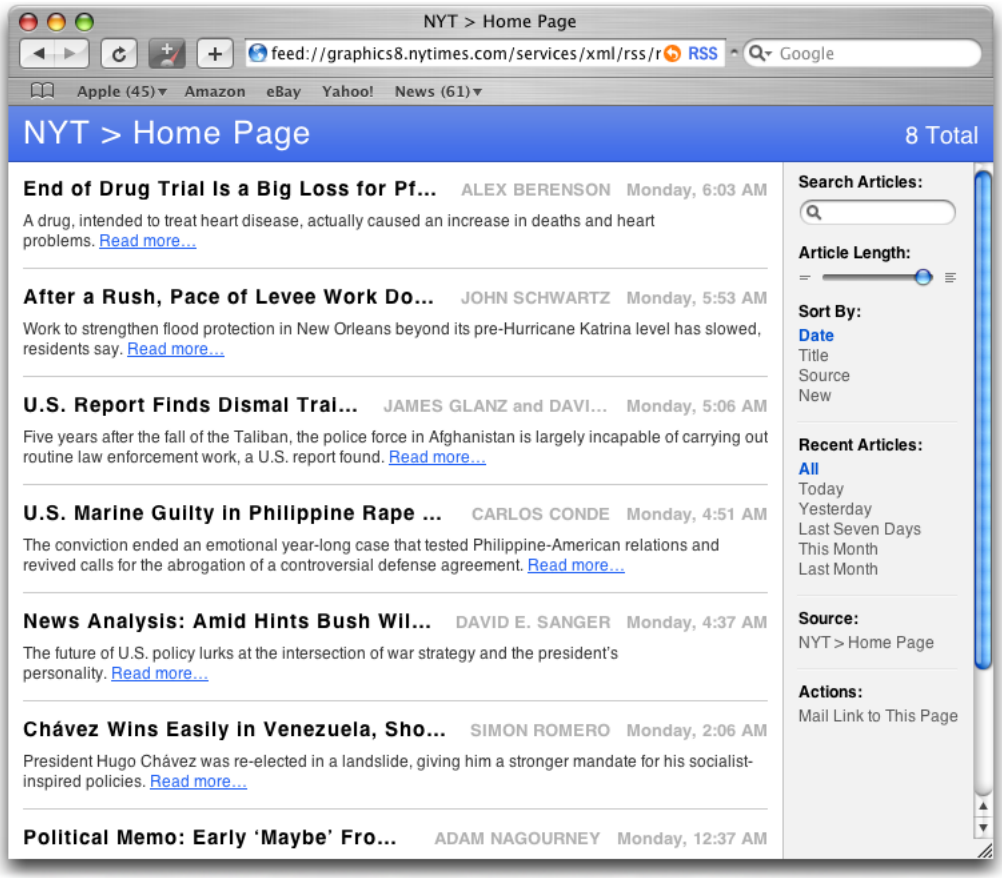
There's a good chance you've already used a feed, even if you didn't know it. When you run Safari in Mac OS X v10.4 Tiger, there are a number of preinstalled bookmarks in the bookmarks bar. In the News folder (in Figure 1-1), many of the bookmarks, such as the Washington Post and CNET News, are feeds, not webpages. Every 30 minutes, Safari downloads each of these feeds and checks for any new headlines. If there are new headlines, Safari places a number next to the bookmark corresponding to the number of new entries (Figure 1-1). Safari also shows you the article if you choose the bookmark.

Figure 1-1 Feed bookmarks in Safari



News headlines are usually stored as a feed. When a new headline is available, it is added to the feed as an **entry**. Then, an application on the user's system (such as Safari) downloads the updated feed, parses it, and checks for the new headline. Figure 1-2 shows how Safari displays a feed of the New York Times entries.

Figure 1-2 A feed viewed in Safari



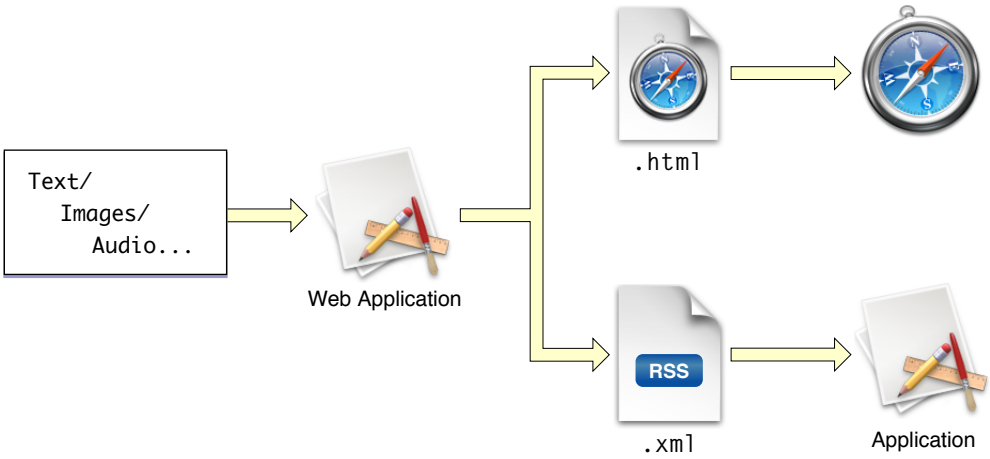
Feeds are not limited to text; you can make a feed that links to any type of data. For example, a podcast is simply a feed with audio files in addition to text. Similarly, a photocast is a feed with images.

Feeds can also be used for publishing any binary data. For instance, you could create a feed that contains software updates for your application. Your application can check the feed and, when a new update is available, download it from the server.

How Feeds Are Generated

Since feeds are simply XML documents, they can be created using a text editor. Adding new entries by hand, however, is a tedious and error-prone process. Typically, the life of a feed begins when the user creates an entry (text, graphics, audio, and the like) and adds it to a database. Then an application takes the entries from the database and produces a feed. Often the application creates more than just a feed; it also generates a webpage. The application that generates the feed can be either local to the user's system or web-based. See Figure 1-3.

Figure 1-3 Feed generation workflow



Feed Formats

A feed format is a specific set of XML **elements** used in a feed. Publication Subscription supports four commonly used feed formats:

- RSS 0.9
- RSS 1.0
- RSS 2.0
- Atom Syndication Format

Even though these standards have similar names, their elements are very different. For example, compare the XML elements for an entry in RSS 2.0 format (Listing 1-1) with the elements in the Atom Syndication Format (Listing 1-2). You'll notice that while the content of both entries is the same, the elements that define them are different.

Listing 1-1 A feed entry in RSS 2.0

```
<item>
    <title>Welcome!</title>
    <pubdate>Fri, 27 Oct 2006 18:51:39 GMT</pubdate>
    <author>Matt</author>
    <description>Hello World!</description>
</item>
```

Listing 1-2 A feed entry in the Atom Syndication Format

```
<entry>
    <author>
        <name>Matt</name>
    </author>
    <title>Welcome!</title>
    <modified>2006-10-27T18:51:39Z</modified>
    <issued>2006-10-27T18:51:39Z</issued>
```

```
    <content type="text/plain">Hello World!</content>
</entry>
```

Publication Subscription is designed to interpret each of these four formats. Since the API encapsulates each of the formats, no matter which format your feed is in, the methods to interpret them are the same.

For more information about each of the XML formats:

- Read http://en.wikipedia.org/wiki/Really_Simple_Syndication to learn more about the RSS 0.9, RSS 1.0 and RSS 2.0 formats
- Read http://en.wikipedia.org/wiki/Atom_%28standard%29 to learn more about the Atom Syndication Format.

What Is Extension XML?

There may be times when the feed you subscribe to uses elements that are not part of one of the feed formats. **Extension XML** extends the specifications to support application-specific data or objects, similar to a plug-in system for feeds. Extension XML refers to a collection of elements outside of the feed format, also known as **namespace**. Each namespace is identified by a unique URL. Namespace URLs are defined at the beginning of a feed, and are often associated with an easier to remember string. This string is known as a **namespace prefix**.

For example, a feed that has bank account information might use a bank namespace identified by the URL `http://www.example.com/bank`, and a namespace prefix of `bank`. This namespace might have elements such as `owner`, `address`, `checking` and `savings`. An entry using this namespace would look like the one in Listing 1-3.

Listing 1-3 An Atom extension XML example

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:bank="http://www.example.com/bank">

...

<item>
    <title>Account2004</title>
    <bank:owner>John Doe</bank:owner>
    <bank:address>1 Infinite Loop, Cupertino, CA 95014</bank:address>
    <bank:checking>111829384</bank:checking>
    <bank:savings>949289291</bank:savings>
</item>
```

There are many additional namespaces already defined. One of the most popular ones if for an iTunes podcast. A good resource for finding namespaces is The Dublin Core Metadata Initiative, as well as rss-extensions.org.

Understanding the structure and workflow of a feed is important to using the Publication Subscription framework. Knowing about the components of a feed and the organization of the four major feed standards will help you understand the organization of the Publication Subscription framework and its components.

Publication Subscription Overview

When used properly, Publication Subscription allows your application to subscribe to feeds, and update the feeds transparently. However, the technology requires the cooperation of multiple components. This chapter describes the major components of Publication Subscription, along with the architecture of the framework.

Publication Subscription Components

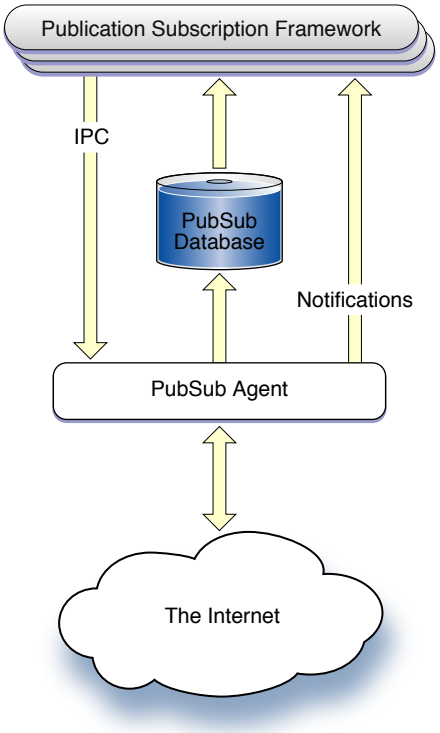
The Publication Subscription technology provides an architecture for your application to subscribe feeds. There are three major components to the technology: the Publication Subscription framework (`PubSub.framework`), the PubSub Agent, and the PubSub Database.

The framework is a collection of Objective-C classes that provide an abstraction for each piece of a feed. The framework uses a background application to perform such tasks as downloading feeds from the Internet, and updating feeds. This background application, called the PubSub Agent, is also responsible for sending notifications to clients of the Publication Subscription framework, and receiving interprocess communications (IPCs) from the clients when necessary.

When feeds are downloaded, the PubSub Agent stores the new information into a database known as the PubSub Database. For each user account there is a single PubSub Database, so a feed is only stored once per user account. If two separate applications are subscribing to the same feed, only one copy is stored in the database.

The Publication Subscription components can be seen in Figure 2-1.

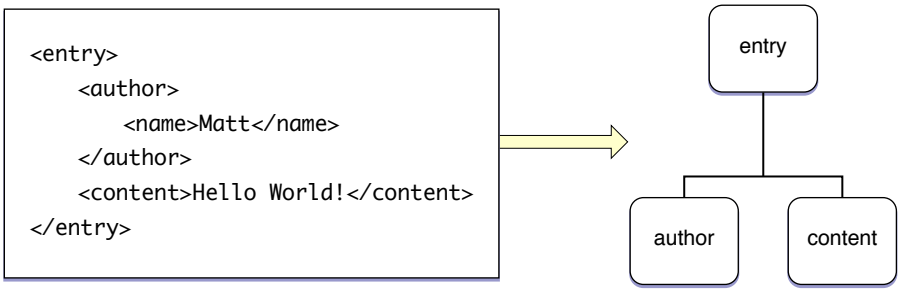
Figure 2-1 Publication Subscription layers



Publication Subscription Framework

The Publication Subscription framework is designed to have a similar structure to that of a set of feeds. Each component of a feed (including the feed itself) is stored as an object, and the object's hierarchy mimics that of a feed (see Figure 2-2). At the same time, the Publication Subscription framework follows the observer design pattern. The observer design pattern defines a one-to-many dependency between objects so that when the subject object changes state, all its dependents are notified and updated automatically. (For more information about the observer design pattern, read Cocoa Design Patterns.)

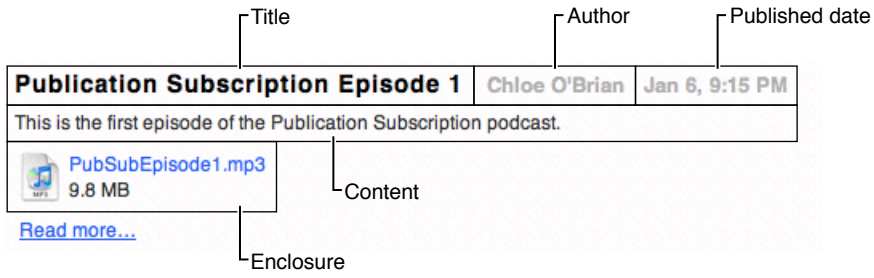
Figure 2-2 A feed structure as a set of objects



For your application to communicate with the PubSub Agent, it needs to register itself with Publication Subscription by generating a client object (`PSClient`). Each application that uses Publication Subscription needs its own client object. If you want to subscribe to a feed, you need to tell the client object. Similarly, if there is a change in a feed your application subscribes to, the client notifies your application. In terms of the observer model, the subject object of the Publication Subscription framework is the client object.

The client object maintains a set of feed objects, one for each feed that it is subscribed to. A feed object (`PSFeed`) stores information about a feed, such as its title, its URL, the time when it was last updated, and the entries associated with it. Just as a feed contains a number of entries, feed objects contain a number of entry objects (`PSEntry`). An entry object contains the content (`PSContent`), the author (`PSAuthor`), and (if necessary) the enclosure (`PSEnclosure`) of the entry. Figure 2-3 shows an entry, and each of its components, viewed using Safari.

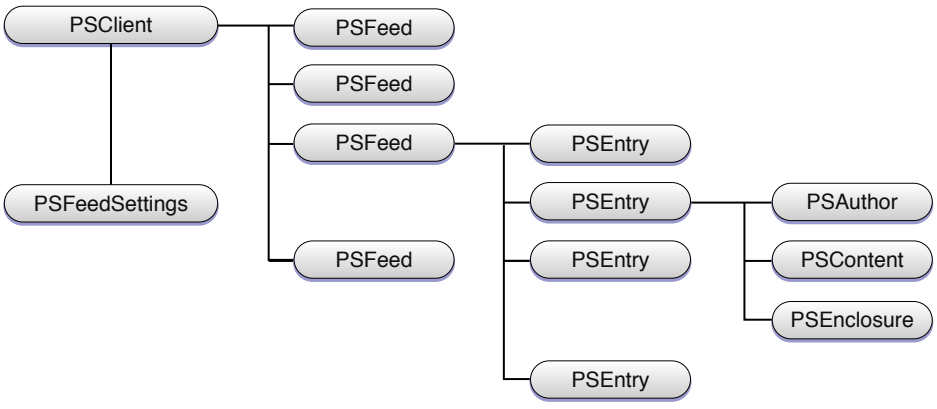
Figure 2-3 An entry viewed with Safari



You can adjust the settings for each client and feed object. For example, you can specify how often to check for feed updates or how long entries should be stored in the PubSub Database. You specify the settings with a settings object (`PSFeedSettings`). A settings object can be associated with either a client or a feed.

The structure of the subscription objects can be seen in Figure 2-4.

Figure 2-4 Subscription object structure



Understanding the components of Publication Subscription and the Publication Subscription framework is necessary for subscribing to feeds within your application. The next two chapters explain how to use the Publication Subscription framework in an application.

Subscribing to a Feed

To view a feed, your application needs to either register with the PubSub Agent and subscribe to the feed, or download the feed and use the Publication Subscription framework to parse it. This chapter describes how to perform both of these tasks, as well as how to adjust the preferences for retrieving the feed.

Generating a Client

To register your application with Publication Subscription, create a client object. Sending `applicationClient` to the `PSClient` class returns a client object for the current application, as the following code shows.

```
PSClient *client = [PSClient applicationClient];
```

You can also create a client object to inspect feeds of another application. To create this client object, you need the bundle identifier of the other application. The bundle identifier is a string using the reverse-DNS naming convention that uniquely identifies each application that uses Publication Subscription. Rather than using the `applicationClient` method to create the new client object, send `clientForBundleIdentifier:` to the `PSClient` class and pass the bundle identifier.

For example, for your application to view the feeds that Mail subscribes to, you would use:

```
PSClient *mailClient = [PSClient clientForBundleIdentifier:@"com.apple.mail"];
```

Using a Subscribed Feed

After your application has its client object, it can subscribe to feeds. First, create an `NSURL` object containing the URL to the feed. Then, use the client object method `addFeedWithURL:` to subscribe the client to the feed and return a newly initialized `PSFeed` object. The code should look like Listing 3-1. Keep in mind that the `addFeedWithURL:` method does not create duplicate entries in the database. If you add the same feed twice, you receive the same data from the original database entry.

Listing 3-1 Subscribing a client to a feed

```
NSURL    *url    = [NSURL URLWithString:
                        @"http://www.apple.com/main/rss/hotnews/hotnews.rss"];
PSFeed   *feed   = [client addFeedWithURL:url];
```

Some feeds may require a user name and password for users to access them through HTTP authentication. Store the authorization information with the feed by setting the `login` property of the feed object for the user name, and use the `setPassword:` method to store the password in the user's default keychain. Publication Subscription also uses the same cookie database as Safari, so if the user is logged into the site of the feed through Safari, Publication Subscription also has access. If you don't store the authorization

information with the feed and the user is not logged in through Safari, the PubSub Agent requests the authorization information from the user. Listing 3-2 shows how to store the user name and password in the feed object.

Listing 3-2 Accessing a protected feed

```
// Place user name and password in NSString objects
NSString *login = @"username";
NSString *password = @"password";

// Store the authorization information in the feed
feed.login = username;
[feed setPassword: password];
```

Using a Feed Without a Subscription

You can create a feed object without subscribing to the feed. In this situation, you need to use the feed object method `initWithURL:` and pass the URL of the feed. See Listing 3-3.

Listing 3-3 Initializing a `PSFeed` object without a subscription

```
NSURL    *url    = [NSURL URLWithString:
                        @"http://www.apple.com/main/rss/hotnews/hotnews.rss"];
PSFeed   *feed   = [[PSFeed alloc] initWithURL:url];
NSError *error;
[feed refresh:&error];
```

If you do this, the feed is not automatically updated as a subscribed feed would be. Instead, you are using the Publication Subscription framework to parse the feed.

Storing a Feed's Preferences

A client can subscribe to multiple feeds, but not all feeds are updated at the same interval. You may have one feed that needs to be checked every five minutes, while another feed only needs to be updated every three hours. Similarly, you may want the time it takes an entry to expire to depend on the feed. The feed settings objects allow you to customize the preferences for each feed. Each feed object is instantiated with a group of default settings.

To customize the preferences for a feed, obtain a feed settings object by either using the `settings` property of a feed object, or the `PSFeedSettings defaultFeedSettings` class method. Then adjust the settings as you see fit with the different properties. Store the updated feed settings in the feed object.

Listing 3-4 retrieves the settings object, sets the entries to expire after 30 days, sets the feed to update every 30 minutes, and then puts the settings object back in the feed.

Listing 3-4 Modifying the settings of a feed

```
// Retrieve PSFeedSettings object from feed
PSFeedSettings *feedSettings = feed.settings;
```

```
// Set entries to expire after 30 days (60 s/m * 60m/h * 24 h/d * 30 d)
feedSettings.expirationInterval = 2592000;

// Set feed to check for updates every 30 minutes (60 s/m * 30m)
feedSettings.refreshInterval = 1800;

// Store settings back in the feed
feed.settings = feedSettings;
```

You can also set the feed settings for a client. If you do, a feed that uses the default settings inherits the client's settings instead of the default ones. However, any settings unique to a feed overrides the settings of the client.

You now know how to create a feed object for any web feed and adjust the preferences for them. The next chapter, "Viewing and Retrieving Content," explains how to obtain the entries and other information in the feed.

Viewing and Retrieving Content

After you obtain a feed object, you need to be able to retrieve the information from within it. Sometimes this information contains not only text, but also files in enclosures or extension XML. The following chapter describes what information is available in an entry and how to access it.

Where's My Entry?

Most feeds contains one or more entries. To retrieve the entries from a feed object, use either the `entries` property (for an array of entry objects) or the `entryEnumeratorSortedBy:` method (for an enumerator of entry objects).

An entry object contains all the information specific to an entry, such as its title, its URL, its authors, its content, and, if necessary, its enclosures. Most of this information is a simple string or URL. However, three of components are a little more complicated.

In some feed standards, the author contains not only a name but also contains an email address and a homepage (see "Feed Formats" (page 11)). The author information in Publication Subscription is stored in an author object. An author object represents a name, an email address, a homepage and a link to an `ABPerson` object.

The content of an entry is also more complex than a simple string. It often comes in one of two different forms: either plain text, or HTML formatted text. In Publication Subscription the content and the summary of an entry are stored as content objects. A content object contains methods for retrieving the content either as a plain text string or an HTML string. The content is returned in the specified format based on which accessor method is used.

An enclosure is a way to attach a file to an entry. If an entry contains enclosures, they will be linked to the entry object. For more information about enclosures, read "Downloading Enclosures" (page 22).

To print out the title and content of every entry in a feed, the code would look like Listing 4-1.

Listing 4-1 Accessing every entry in a feed

```
// Retrieve the entries as an unsorted enumerator
NSEnumerator *entries = [feed entryEnumeratorSortedBy: nil];
PSEntry *entry;

// Go through each entry and print out the title, authors, and content
while (entry = [entries nextObject]) {
    NSLog(@"Entry Title:%@", entry.title);
    NSLog(@"Entry Authors:%@", entry.authorsForDisplay);
    NSLog(@"Entry Content:%@", entry.content.plainTextString);
}
```

Since it follows an observer design pattern, Publication Subscription can also notify your application when any changes occur to a feed. Register with the notification center to be alerted when any change occurs (`PSFeedEntriesChangedNotification`). When your callback method is invoked, the changed entries are stored as a key-value pair in the user information dictionary of the notification.

Downloading Enclosures

Some entries may contain links to files as enclosures. Publication Subscription provides features that make it easy to download the files within the enclosure. Each enclosure is stored as a enclosure object in its associated entry object. Because there may be more than one enclosure in an entry, the entry object property `enclosures` returns an array of enclosure objects. In most cases, this array only contains one object. Each of these objects contains information about the enclosure's size, URL and MIME type.

By default, enclosures are not automatically downloaded. You can change this setting on a per-feed basis so that any enclosure from a subscribed feed is downloaded with the entry. Listing 4-2 shows how to make a feed download its enclosures automatically.

Listing 4-2 Downloading enclosures automatically

```
PSFeedSettings *settings = feed.settings;
settings.downloadsEnclosures = YES;
feed.settings = settings;
```

If you want to download the file in the enclosures individually, send `download:` to the appropriate enclosure object. The `download:` method is asynchronous, so it rarely returns an error. Instead, check on the status of the download with the `downloadState` property. There are six possible states:

```
PSEnclosureDownloadDidFail
PSEnclosureDownloadDidFinish
PSEnclosureDownloadIsIdle
PSEnclosureDownloadIsQueued
PSEnclosureDownloadIsActive
PSEnclosureDownloadWasDeleted
```

If the download failed, see what caused the failure by using the `downloadError` method. If the download is still active, you can check on its progress by using the `downloadProgress` method. Assuming the download finishes, the location of the downloaded file is available with the `downloadedPath` property.

Although the download status can be checked in a synchronous manner, it is recommended that you register for the notification `PSEnclosureDownloadStateDidChangeNotification` instead. When your callback method is invoked, you can determine the status of the download. Listing 4-3 shows how to start downloading the file in the enclosure and register for the appropriate notification. Listing 4-4 shows a callback method for the notification.

Listing 4-3 Downloading an enclosure

```
// Get the enclosures from the current entry, and retrieve the first one
NSArray *enclosureArray = entry.enclosures;
enclosure = [enclosureArray objectAtIndex: 0];
NSError *error;
```

```
// Download the enclosure
if (![enclosure download:&error]) {
    NSLog(@"Enclosure download failed: %@", error)
} else {

    // Register for any changes to the download's state
    [[NSNotificationCenter defaultCenter]
            addObserver:self
            selector:@selector(downloadStateChanged:)
            name: PSEnclosureDownloadStateDidChangeNotification
            object: enclosure];
}
```

Listing 4-4 Receiving download state changes through notifications

```
- (void) downloadStateChanged: (NSNotification *) sender {

    // See what state change cause the notification to be sent
    switch (enclosure.downloadState) {

        // If the download failed, log why and stop receiving notifications
        case PSEnclosureDownloadStateDidFail:
            NSLog(@"Enclosure download failed: %@", enclosure.downloadError);
            [[NSNotificationCenter defaultCenter]
                removeObserver: self
                name: PSEnclosureDownloadStateDidChangeNotification
                object: enclosure];
            break;

        // If the download succeeded, log the location of the file and stop
        // receiving notifications
        case PSEnclosureDownloadStateDidFinish:
            NSLog(@"Location of downloaded file is: %@",
enclosure.downloadedPath);
            [[NSNotificationCenter defaultCenter]
                removeObserver: self
                name: PSEnclosureDownloadStateDidChangeNotification
                object: enclosure];
            break;

        case default:
            break;
    }
}
```

Extension XML

Many feeds may also contain extension XML elements. If additional namespaces are used in a feed, use the `extensionXMLElementsUsingNamespace:` method to return an array of `NSXMLElement` objects. Pass the namespace URL, not the prefix, to the `extensionXMLElementsUsingNamespace:` method. If your feed had an entry like that in Listing 1-3 (page 12), `extensionXMLElementsUsingNamespace:` returns an array of four `NSXMLElement` objects.

Listing 4-5 shows how to find a specific element in a particular namespace in an entry. In this example, the namespace is the iTunes Podcast.

Listing 4-5 Retrieving extension XML from an entry

```
// Find each element using the iTunes Podcast namespace
for( NSXMLElement *elem in [entry extensionXMLElementsUsingNamespace:
@"http://www.itunes.com/dtds/podcast-1.0.dtd"] ) {

    // Check if the element is called "keywords"
    if (NSOrderedSame == [[elem localName] isEqualToString:@"keywords"]) {

        // If it is, print the data from the element to the log
        NSLog(@"keywords:%@", [elem stringValue]);
    }
}
```

Glossary

access control list (ACL) A list of users or groups. Each user or group has associated permissions, such as allow or deny. ACLs are used to control access to files, folders, and services.

content The text or data in an entry. The content is placed in a `<description>` element for RSS, or a `<content>` element for Atom.

element An element is the information in an XML document that begins with a start-tag (`<element>`) and ends with an end-tag (`</element>`). Elements can contain sub-elements, to create a tree-like structure.

enclosure An element used for attaching files to an entry. An enclosure element contains a link to a single file, along with the file's MIME type and size. Enclosures are used to link to audio files in podcasts, or images for photocasts.

entry An element that can contain the following sub-elements: author, date , description, and enclosure.

extension XML The use of additional namespaces in an XML document.

feed An XML document that conforms to one of the four major feed formats: RSS 0.9, RSS 1.0, RSS 2.0 or Atom.

namespace A group of elements, each used for a specific purpose. Namespaces are used to distinguishing between elements of the same name.

namespace prefix A shorter name used to refer to a namespace in an XML document. A namespace prefix is designated at the beginning of an XML document using the `feed` attribute `xmlns:prefix="http://www.example.com/namespace"`.

Document Revision History

This table describes the changes to *Publication Subscription Programming Guide*.

Date	Notes
2007-05-11	New document describing the framework for subscribing to RSS and Atom feeds.

Document Revision History