iPod Library Access Programming Guide

Audio & Video: Audio

2010-05-21

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

Simultaneously published in the United States and Canada.

Contents

Figures and Listings

Introduction

iPod library access lets your application play a user's songs, audio books, and audio podcasts. The API design makes basic playback very simple while also supporting advanced searching and playback control.

iPod library access opens up iOS to allow a wide range of music-related enhancements to your application. Here are just a few ideas for what you can do:

- Let a user assemble and play a soundtrack for your game or exercise app
- Let a user retrieve the names of their favorite artists, songs, and albums from their iPod library to send to a friend
- Provide a recommendation service for new music or artist touring schedules based on the content of a user's iPod library

Before reading this document, you should already be comfortable with iOS development as described in *iOS Application Programming Guide* and *iOS Development Guide*. For a tutorial introduction to creating programs for iOS, read *Your First iOS Application*.

iPod Library Access Guide complements *Media Player Framework Reference*, which you may find helpful to refer to as you read this document.

Note: iPod library access works only on devices and not in the Simulator. This is because the Simulator has no access to a device's iPod library. To develop applications using the classes in this technology, you need a provisioned iOS device.

Organization of This Document

This document includes the following chapters:

- "About iPod Library Access" (page 9)—Provides a complete overview of music playback and iPod library access using this API.
- "Using Media Playback" (page 19)—Explains how to create and use music players.
- "Using the Media Item Picker" (page 23)—Shows how to invoke the picker and implement its delegate methods for retrieving the media items chosen by a user.
- "Using the iPod Library" (page 25)—Goes into depth on creating and using predicates and queries to retrieve media items from the iPod library.
- "Glossary" (page 33)—Defines terminology used in describing this API.

See Also

To get the most out of this technology, take advantage of these resources:

- *Media Player Framework Reference*—complete reference documentation for the classes introduced in this document.
- *AddMusic*—an iPhone sample project that you can download, study, adapt, and extend for use in your iOS applications.
- *Audio Session Programming Guide*—explains how to configure the overall audio behavior for your application. Read this document if you are creating an application that uses your own audio along with iPod audio playback.
- *Core Audio Overview*—explains how audio formats can mix in iOS and describes the available audio formats for application audio.

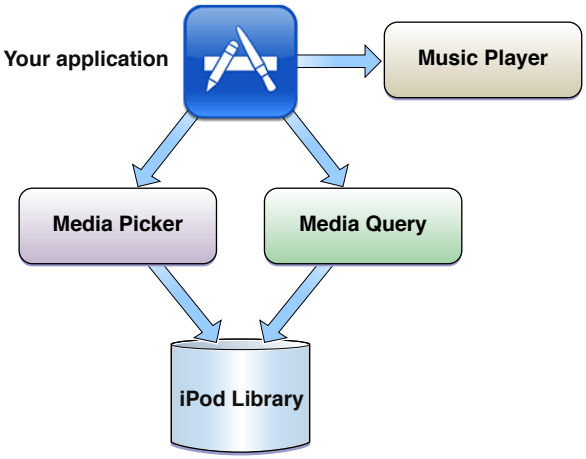
About iPod Library Access

iPod library access is the iOS interface for retrieving and playing items from the iPod library. The iPod library is the set of media items on a device that a user has synced from iTunes on the desktop.

As shown in Figure 1-1, your application has two ways to retrieve items. The media item picker, shown on the left, is an easy-to-use, pre-packaged view controller that behaves like the built-in iPod application's music selection interface. For many applications, this is sufficient.

If the picker doesn't provide the specialized access control that you want, the media query interface—shown to the lower right of your application in the figure—will. It supports predicate-based specification of items from the iPod library.

Figure 1-1 Using iPod library access



As depicted to the right of your application in the figure, you then play the retrieved media items using the music player provided by this API.

As you can see, the classes in this API address two, usually distinct areas of development: database access—for finding media items to play—and music playback. However, you do not need to be a subject expert in either area to use this API effectively. iPod library access does the "heavy lifting" for you.

For example, the entirety of your music playback code is a single line—whether playing DRM-encrypted AAC files, Apple Lossless tracks ripped from a CD, audio podcasts encoded with iLBC, or audio books. The system automatically sets up buffering, chooses the appropriate codec, and sends the audio directly to the device output hardware.

Using a music player and the media item picker, you can implement music selection and playback without writing any code to access the iPod library. The database access classes provide a complete query system when you need it, and stay out of your way when you don't.

Note: iPod library access applies only to audio-based media items. You cannot play video podcasts, movies, or television shows from the iPod library.

About Music Players

Your application uses a music player in a way that's similar to an end user operating the built-in iPod application. You can programmatically play, pause, seek, and so on.

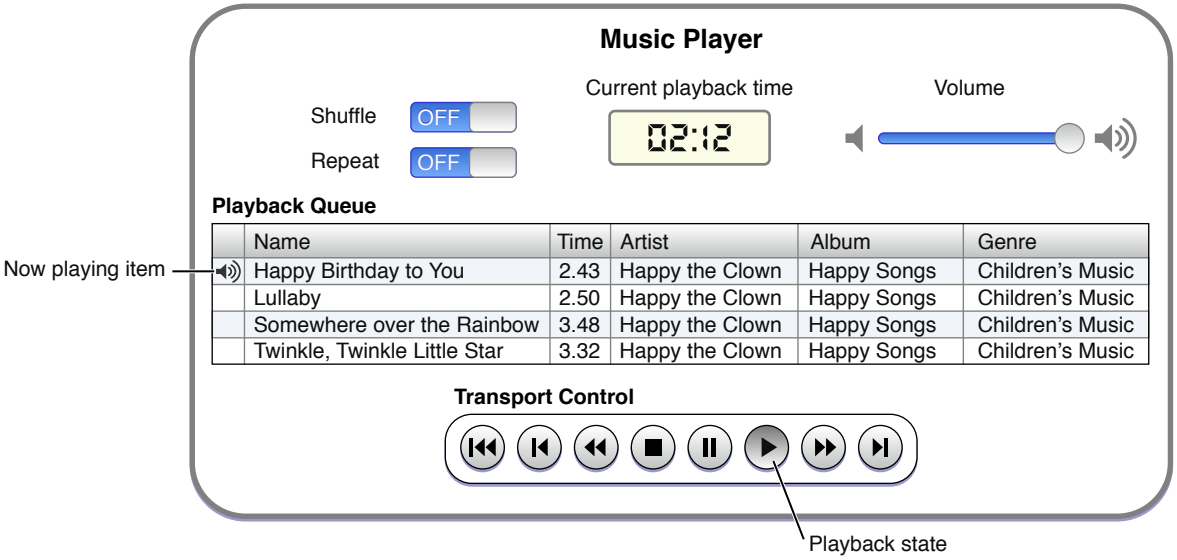
Music Player Basics and Terminology

A **music player** is the object you use for playing media items. It has a **playback queue**, which is a list of media items to play. A **media item** is a song, audio podcast, or audio book.

Media items get onto a device when a user syncs them from iTunes on the desktop. On the device, the complete set of media items is called the **iPod library**.

A music player knows which item is now playing or designated to play, and knows where in that item's timeline playback is at a given moment. These attributes are depicted schematically in Figure 1-2. (The figure is for explanation only. A music player does not provide a user interface.)

Figure 1-2 Schematic representation of a music player object



The **now playing** item has special status. For example, if a user pauses a song in the built-in iPod application, and then launches your application, your music player can continue playing that item from the same spot.

Additional properties round out the music player, making it highly flexible. As shown in the figure, a music player also has modes, playback state, and volume.

- **Shuffle mode** and **repeat mode** operate the same way they do in the built-in iPod app.

- **Playback state** is what you'd expect from any audio playback system: playing, stopped, paused, fast forward, or rewind. In addition, you can skip to the beginning of the next or previous media item and return to the start of a playback queue.
- **Volume** is full by default and can be set to any value down to silent.

You can obtain two flavors of music player, depending on the goals of your application.

- The **application music player** plays music locally within your application. It is not influenced by, nor does it affect, the state of the built-in iPod application. Specifically, your music player has a different now-playing item, playback state, and modes. When a user quits your application, music that you are playing stops.
- The **iPod music player**, in effect, employs the built-in iPod app on your behalf; it shares the iPod's state. When a user quits an app that is using this player, the music continues to play.

Only one music player can play audio at a time.

Hello Music Player

Here is a bare bones *hello world*-style example that demonstrates library access and music playback. In a few minutes you can have a working, if minimal, music player. Lacking a user interface, this code queues up the entire iPod library and starts playing immediately on launch.

Note: To follow these steps you'll need a provisioned device because the Simulator has no access to a device's iPod library.

1. **Create a new Xcode project.**

   In Xcode, create a new project using the Window-based Application template. In the project window, add the `MediaPlayer` framework to the Frameworks group. Save the project.

2. **Import the umbrella header file for the Media Player framework.**

   Add the following line to the `AppDelegate.h` file, after the existing `#import` line:

   ```
   #import <MediaPlayer/MediaPlayer.h>
   ```

3. **Add code to create a music player, assign it music to play, and start playback.**

   Open the project's `AppDelegate.m` implementation file. Before the end of the `applicationDidFinishLaunching:` block, add the three lines of code shown in Listing 1-1.

Listing 1-1 A very bare-bones music player

```
// instantiate a music player
MPMusicPlayerController *myPlayer =
    [MPMusicPlayerController applicationMusicPlayer];

// assign a playback queue containing all media items on the device
[myPlayer setQueueWithQuery: [MPMediaQuery songsQuery]];
```

```
// start playing from the beginning of the queue
[myPlayer play];
```

Now, configure your project appropriately for your development device, which includes setting the code-signing identity and the bundle identifier. Also, ensure that the device has at least one song in its iPod library. Build and run the project. When the application launches on the device, the first song in the iPod library starts playing. The player continues to play through all the items in the iPod library or until you quit the application.

About Music Player Change Notifications

To keep track of what a music player is doing, you register for music player change notifications. This is essential for ensuring that your application's state and the music player's state are well coordinated.

For example, here is the sequence of events for correctly starting up music playback. Notice that, because music players operate on their own threads, you do not update your user interface until receiving the appropriate notification.

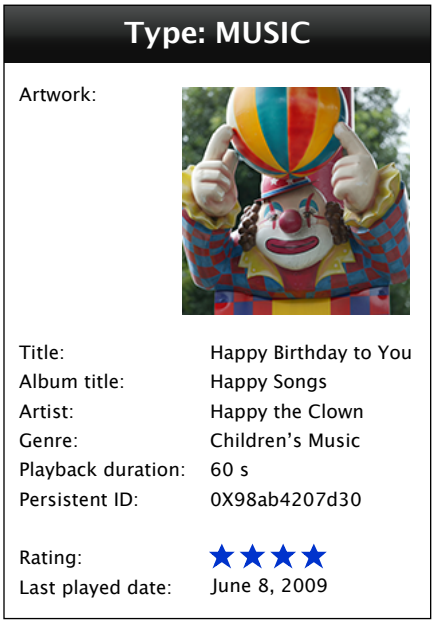
1. A user taps Play.
2. Your application invokes playback on the music player.
3. The music player starts playing and issues a playback-state-change notification.
4. Your application receives the notification and queries the music player's state, confirming that it is indeed playing.
5. Your application updates its user interface accordingly—perhaps changing the Play button to say Pause.

Music player change notifications support keeping track of playback state, the now-playing item, and the music player's playback volume. "Using Media Playback" (page 19) explains how to use them.

About Media Items and the iPod Library

Media items—the songs, audio books, and audio podcasts in the iPod library—can have a wide range of metadata. You can use this metadata in building queries and in creating attractive displays of the media items in the user interface. Figure 1-3 gives you an idea of the character of a media item.

Figure 1-3 A media item and some of its metadata



All media item metadata is read-only. However, by using media item Persistent ID values, you can associate additional metadata that you manage in your application.

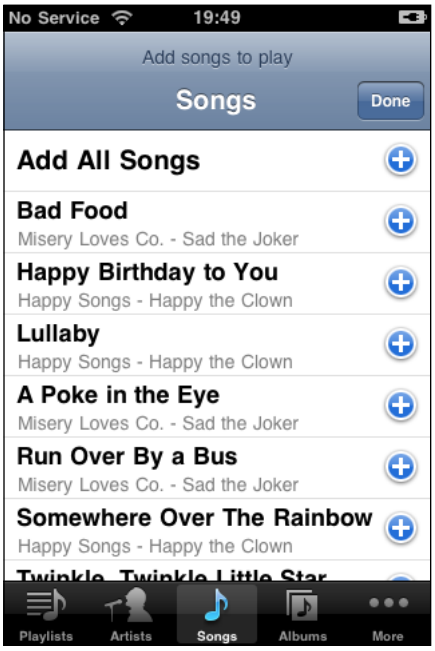
As Figure 1-3 suggests, an item's metadata can include more than one category of information. So-called **general properties** are those that can apply to any media item. These include "title," "artist," "artwork," and many others. The values of these properties typically do not change over time.

A media item can also have **user-defined properties** such as rating and last-played date. These properties update according to user activity, just as they do on the desktop.

The Media Item Picker

The simplest way to enable users to choose music is to use a **media item picker**—a fully-configured modal view controller. Its user interface is similar to the built-in iPod application's on-the-go interface, as shown in Figure 1-4.

Figure 1-4 The user interface of the media item picker



When the user taps Done, a delegate method that you implement receives the list of chosen media items and then dismisses the picker's view.

It's important to know that the similarity between the media item picker and the built-in iPod app is only skin deep. Using the iPod, a user can build an on-the-go playlist. That playlist acts like other playlists; for example, it appears in the Playlists tab of the iPod application and it persists until the user changes or deletes it.

With the picker, a user instead specifies a *collection* of media items. The collection does not have the status of a playlist. It won't persist after the user quits your application—unless you archive it. Neither will the collection appear, under any circumstances, in the Playlists tab of the picker.

Getting Media Items Programmatically

If the media item picker doesn't provide the control you want, you can use the database access classes from this API. These classes are designed to let you create arbitrarily complex queries. You could, for example, retrieve all the songs in a particular genre whose titles include a particular word or phrase.

Using programmatic access is a two step process:

1. Configure a query.
2. Ask the query to retrieve its matching media items.

A **media query** is a description of what to retrieve from the iPod library and how those retrieved items should be arranged. It has two properties to configure:

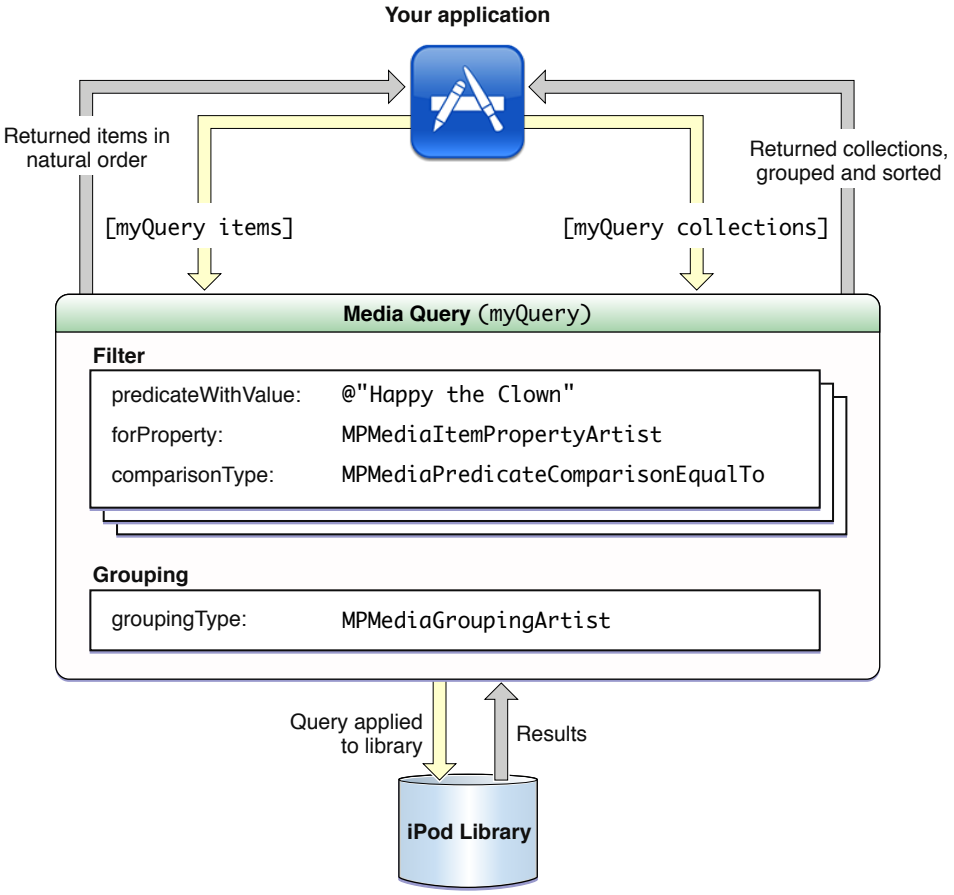
- The **filter** is the description of what to retrieve. The filter is optional; a filterless query matches the entire iPod library.
- The **grouping type** is an optional key that specifies the arrangement to use for retrieved collections of media items.

Zooming in a bit more, the filter can be as simple or complex as your application demands. It consists of one or more instances of a media property predicate. A **media property predicate** is a statement of a logical condition to test each media item against. The items that satisfy the filter are retrieved from the iPod library when you invoke the query.

The optional grouping type specifies the arrangement and sorting of collections as well as the sorting of media items within each collection. For example, using an "album" grouping type results in returned media items grouped by album, with each album's songs sorted in track order.

Figure 1-5 shows a configured media query and its place between your application and the iPod library.

Figure 1-5 Using a media query to access the iPod library



As the figure shows, a query can fetch items or collections.

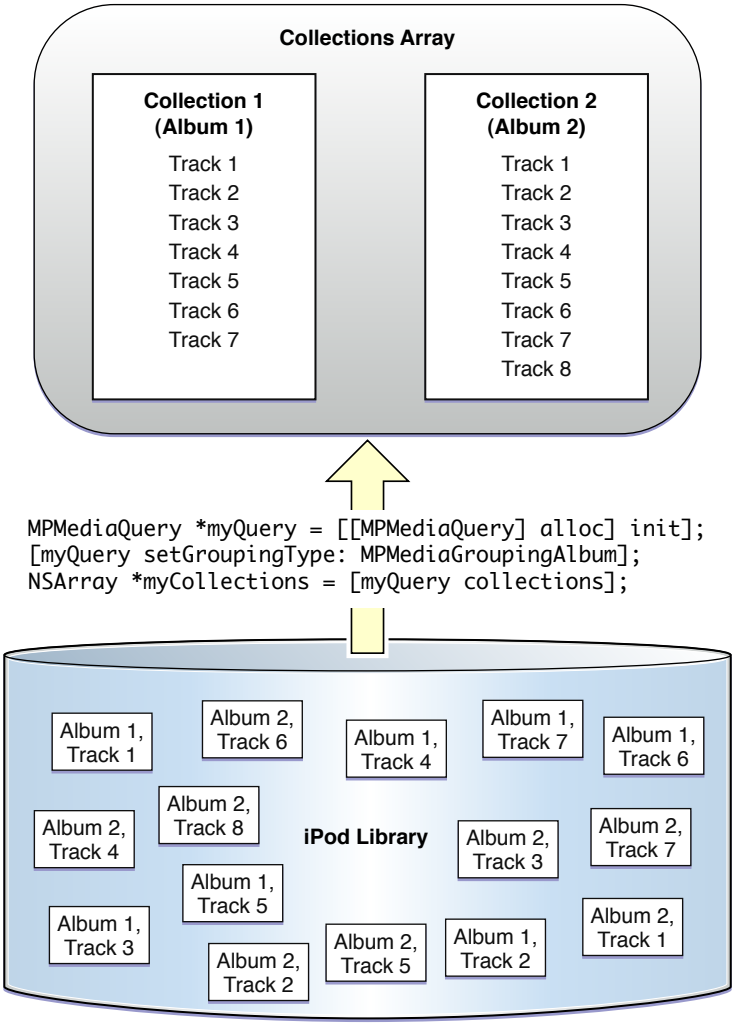
- When you ask for items, the query returns a collection containing all the items that match the filter. The items are in "natural" order, meaning that they are ordered as iTunes shows them on the desktop.
- When you ask for collections, the media query employs not only its filter but also its grouping type.

About Collections and Playlists

A **media item collection** is an array of media items. You obtain collections from the iPod library by accessing a media query's `collections` property. The returned value is a sorted array of collections where each collection is an instance of the query's grouping type.

For example, say you specify an "album" grouping type by assigning the `MPMediaGroupingAlbum` key to a media query—and say that query has no filter. The value of the `collections` property is then the entire audio content of the iPod library arranged as albums, one album per collection. The collections (albums in this case) are sorted alphabetically. The songs (each a media item) within each collection are sorted by track number. Figure 1-6 illustrates this.

Figure 1-6 Obtaining collections from the iPod library



At the bottom of the figure you see the iPod library for a device. In this illustrative case, the library has just two music albums, one with seven tracks and the other with eight. The items are in their "natural" order, as they appear in iTunes on the desktop.

The up-pointing arrow in the middle of the figure represents the definition of a generic (filterless) query, the application of the "album" grouping type, and the accessing of that query's `collections` property.

The top of the figure represents the result of the `collections` call. It is an array whose elements are, in turn, arrays of media items. The collections array is sorted by album name. Each collection is sorted by track number.

You can construct your own collections as well. This can be useful, for example, for managing the selections a user has made with a media item picker. Note, however, that collections are not mutable.

A **playlist** is special sort of media item collection. It has properties that regular collections do not have, such as a name and a persistent ID. Playlists are created by users on the desktop; on the device they are read-only.

Using iPod Library Change Notifications

The `MPMediaLibrary` object represents the state of the iPod library. You can use it to ensure that your cache of the library contents is up to date. This is useful because a user may sync their device, changing the content of the iPod library, while your application is running.

Mixing Your Own Sounds with iPod Library Sounds

Note: To mix application sounds with sounds from the iPod library, you need to understand audio sessions, audio session categories, audio hardware route changes, and audio interruptions. To learn about these, refer to *Audio Session Programming Guide*.

You also need to understand how your choice of audio formats can impact the simultaneous playback of sounds on a device. See "Playing Multiple Sounds Simultaneously" in *Core Audio Overview*.

A music player automatically employs the Media Playback audio session category. If your application uses a music player and no other sounds, you should write no audio session code. Specifically, you should not initialize, configure, or activate the audio session. The system automatically handles playback, audio hardware route changes, and audio interruptions for music players.

On the other hand, to mix application sounds with iPod library sounds, you need to configure and use your application's audio session. Employ the Ambient category to support mixing. Handle audio interruptions, audio hardware route changes, and audio session reactivation as described in *Audio Session Programming Guide*. For an example of mixing application sound with iPod sound, see the *AddMusic* sample.

Using Media Playback

To play songs, audio books, and audio podcasts from a user's iPod library, you use a music player—an instance of the `MPMusicPlayerController` class. In a nutshell, the steps to perform are as follows:

1. Register to receive music player notifications, and turn on notifications
2. Create a music player
3. Set up a playback queue for the music player
4. Configure the playback options
5. Invoke playback

This chapter describes how to implement these steps. It also provides guidance for handling some scenarios you may encounter when using a music player, such as appending to an existing playback queue while music is playing. There are expanded versions of the code examples from this chapter in the *AddMusic* sample application.

Registering for Music Player Notifications

If you provide music player playback control, or if you display information about the now-playing item, you must register for music player notifications as shown in Listing 2-1.

Listing 2-1 Registering for and activating music player notifications

```
NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];

[notificationCenter
    addObserver: self
    selector:    @selector (handle_NowPlayingItemChanged:)
    name:        MPMusicPlayerControllerNowPlayingItemDidChangeNotification
    object:      musicPlayer];

[notificationCenter
    addObserver: self
    selector:    @selector (handle_PlaybackStateChanged:)
    name:        MPMusicPlayerControllerPlaybackStateDidChangeNotification
    object:      musicPlayer];

[musicPlayer beginGeneratingPlaybackNotifications];
```

You implement the methods provided to the *`selector:`* parameters shown here. For example implementations, see the *AddMusic* sample.

Before deallocating a music player, unregister for notifications and then turn them off as shown in Listing 2-2.

Listing 2-2 Unregistering and deactivating music player notifications

```
[[NSNotificationCenter defaultCenter]
    removeObserver: self
    name:           MPMusicPlayerControllerNowPlayingItemDidChangeNotification
    object:         musicPlayer];

[[NSNotificationCenter defaultCenter]
    removeObserver: self
    name:           MPMusicPlayerControllerPlaybackStateDidChangeNotification
    object:         musicPlayer];

[musicPlayer endGeneratingPlaybackNotifications];
```

Creating and Configuring a Music Player

To create an application music player, just call the `applicationMusicPlayer` class method as shown in the first line of Listing 2-3.

Listing 2-3 Creating an application music player

```
MPMusicPlayerController* appMusicPlayer =
    [MPMusicPlayerController applicationMusicPlayer];

[appMusicPlayer setShuffleMode: MPMusicShuffleModeOff];
[appMusicPlayer setRepeatMode: MPMusicRepeatModeNone];
```

As the listing shows, you may also want to configure shuffle and repeat modes so that they do not take on the iPod app's modes—which they would do by default.

The various shuffle and repeat modes are described in *MPMusicPlayerController Class Reference*.

Creating an iPod music player has an extra step because you have access to the built-in iPod application's state and there may be a now-playing item as your application launches. Listing 2-4 shows how to test for a now-playing item.

Listing 2-4 Creating an iPod music player

```
MPMusicPlayerController* iPodMusicPlayer =
    [MPMusicPlayerController iPodMusicPlayer];

if ([iPodMusicPlayer nowPlayingItem]) {
    // Update the UI (artwork, song name, volume indicator, etc.)
    //        to reflect the iPod state
}
```

As shown in the listing, your application can update its user interface and state based on the now-playing item.

Setting Up a Playback Queue

There are two main ways to set up a playback queue for a music player:

- Use a media item collection
- Use a media query, which implicitly defines a collection

You get a collection by using the database access classes or by employing the media item picker. Either way, applying the collection to a music player involves a single method call, shown here:

```
[musicPlayer setQueueWithItemCollection: userMediaItemCollection];
```

Alternatively, you can set up a queue using a media query—which, in turn, implicitly defines a collection, as shown here:

```
[musicPlayer setQueueWithQuery: [MPMediaQuery songsQuery]];
```

You can, of course, specify a more complex query as an argument to the `setQueueWithQuery:` method. For more on queries, see "Using the iPod Library" (page 25).

Listing 2-5 shows a practical playback queue setup—one that accounts for playback state, whether a new collection should replace the player's queue or append to it, and so on.

Listing 2-5 Setting up a playback queue in context

```
- (void) updateQueueWithCollection: (MPMediaItemCollection *) collection {

    // Add 'collection' to the music player's playback queue, but only if
    //    the user chose at least one song to play.
    if (collection) {

        // If there's no playback queue yet...
        if (userMediaItemCollection == nil) {
            [self setUserMediaItemCollection: collection];
            [musicPlayer setQueueWithItemCollection: userMediaItemCollection];
            [musicPlayer play];

        // Obtain the music player's state so it can be restored after
        //    updating the playback queue.
        } else {
            BOOL wasPlaying = NO;
            if (musicPlayer.playbackState == MPMusicPlaybackStatePlaying) {
                wasPlaying = YES;
            }

            // Save the now-playing item and its current playback time.
            MPMediaItem *nowPlayingItem        = musicPlayer.nowPlayingItem;
          NSTimeInterval currentPlaybackTime = musicPlayer.currentPlaybackTime;

            // Combine the previously-existing media item collection with
            //    the new one
            NSMutableArray *combinedMediaItems =
                [[userMediaItemCollection items] mutableCopy];
            NSArray *newMediaItems = [mediaItemCollection items];
            [combinedMediaItems addObjectsFromArray: newMediaItems];
```

```
            [self setUserMediaItemCollection:
                [MPMediaItemCollection collectionWithItems:
                    (NSArray *) combinedMediaItems]];

            [musicPlayer setQueueWithItemCollection: userMediaItemCollection];

            // Restore the now-playing item and its current playback time.
            musicPlayer.nowPlayingItem      = nowPlayingItem;
            musicPlayer.currentPlaybackTime = currentPlaybackTime;

            if (wasPlaying) {
                [musicPlayer play];
            }
        }
    }
}
```

This method branches according to whether or not there is a preexisting playback queue, and ensures that the playback state is preserved when appending to a playback queue.

Controlling Playback

Playback control for music players is straightforward, as described in *MPMusicPlayerController Class Reference*. You can play, pause, stop, seek forward and back, and skip forward and back.

In addition, the `currentPlaybackTime` property is read/write; you can use it to set the playback point within the now-playing item's timeline.

For example, you could let the user set the playback point in a media item's timeline by providing a `UISlider` object in the application interface. In the following example, such a slider is assumed to be connected to a `timelineSlider` instance variable.

```
- (IBAction) setTimelinePosition: (id) sender {
    [musicPlayer setCurrentPlaybackTime: [timelineSlider value]];
}
```

Using the Media Item Picker

The media item picker is the pre-packaged view controller for letting a user choose media items from the iPod library. Using the picker is very simple:

1. Designate a controller object as a delegate of the picker.
2. Invoke the picker from the controller.
3. When the user indicates that they are finished, the delegate receives the chosen collection of media items and dismisses the picker.

Setting Up a Media Item Picker Delegate

To set up a controller object as a media item picker delegate, first add the protocol's name in the interface declaration in the controller's header file, as follows:

```
@interface myController : UIViewController <MPMediaPickerControllerDelegate> {
    // interface declaration
}
```

Next, implement the two delegate methods from that protocol. The first method, shown in Listing 3-1, responds to the user having chosen some media items. It dismisses the picker and invokes the controller's playback queue update method.

Listing 3-1 Responding to a new collection of media items from the picker

```
- (void) mediaPicker: (MPMediaPickerController *) mediaPicker
           didPickMediaItems: (MPMediaItemCollection *) collection {

    [self dismissModalViewControllerAnimated: YES];
    [self updatePlayerQueueWithMediaCollection: collection];
}
```

For example code showing how to update a playback queue, see Listing 2-5 (page 21).

The second picker delegate method handles the case of the user tapping Done without having chosen any items to play. See Listing 3-2 for a basic implementation.

Listing 3-2 Responding if the user cancels the picker

```
- (void) mediaPickerDidCancel: (MPMediaPickerController *) mediaPicker {

    [self dismissModalViewControllerAnimated: YES];
}
```

Displaying a Media Item Picker

Listing 3-3 shows how to configure and display a media item picker—including establishing the controller object as the delegate of the picker.

Listing 3-3 Displaying a media item picker

```
MPMediaPickerController *picker =
    [[MPMediaPickerController alloc]
        initWithMediaTypes: MPMediaTypeAnyAudio];                   // 1

[picker setDelegate: self];                                         // 2
[picker setAllowsPickingMultipleItems: YES];                        // 3
picker.prompt =
    NSLocalizedString (@"Add songs to play",
                       "Prompt in media item picker");

[myController presentModalViewController: picker animated: YES];    // 4
[picker release];
```

Here's how this code works:

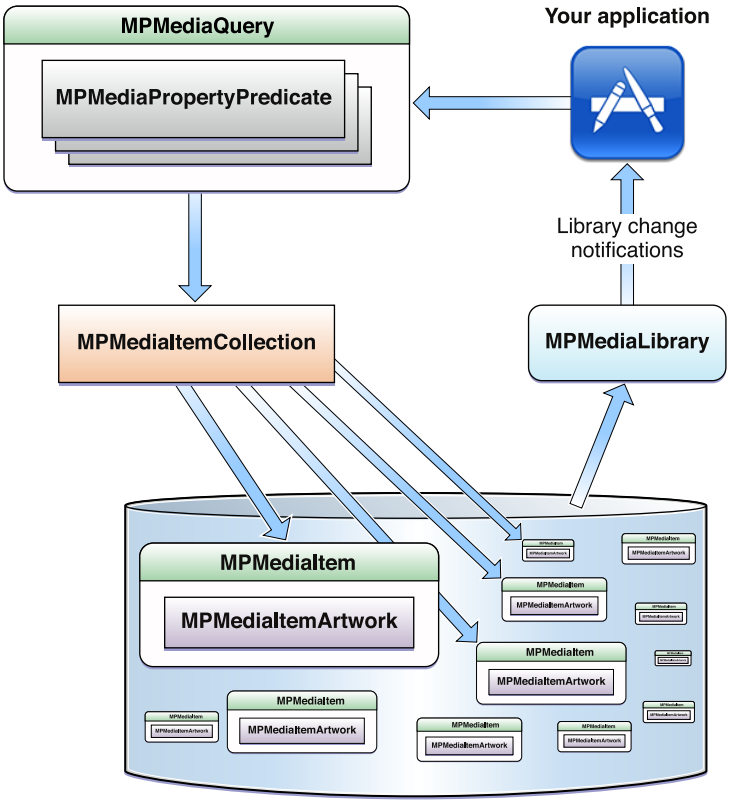
1. Creates a media item picker. The parameter indicates the sort of media items to display. For options, see the `Media Item Type Flags` enumeration.
2. Establishes your controller object as the delegate.
3. Specifies that the user can pick multiple items. You can instead display a single-item picker by excluding this statement; the default behavior is to disallow multiple selection.
4. Displays the picker. The `myController` object retains it so you then release it to balance the `alloc` call.

Using the iPod Library

Your application may need more control over managing and choosing media items than you get with the media item picker. If you want to provide a custom user interface to the iPod library, perform specific queries, or associate custom metadata with media items, you need the database access classes of this API.

Figure 4-1 illustrates how your application and the database access classes interact to retrieve media items.

Figure 4-1 Using the iPod library database access classes



First, take a moment to notice the similarity between this figure and Figure 1-5 (page 15). The earlier figure helped explain what a query is. This figure places the media query class among its cohorts, showing how all the database access classes relate to each other.

The figure illustrates two scenarios of interaction between your application and the iPod library. First, moving counterclockwise from the "Your application" icon, the figure depicts the creation and configuration of a query that in turn defines a media item collection. The collection points to a particular subset of items from the library. Although not shown in the figure, the collection is the return value of invoking the query. Each media item owns a media item artwork object, as shown in the figure, among its other properties.

The second scenario in Figure 4-1 is your application receiving change notifications from the iPod library by way of the `MPMediaLibrary` class. By registering to receive these change notifications, your application can update any cache of library content if a user syncs their device while your application is running.

Retrieving Media Items

Retrieving media items from the iPod library begins with constructing a media query. The simplest query is the generic "`everything`" query shown in Listing 4-1, which matches the entire contents of the library. The example then logs the titles of the media items to the Xcode debugger console, demonstrating use of the `items` property to invoke the query and retrieve the media items it matches.

Listing 4-1 Creating and using a generic media query

```
MPMediaQuery *everything = [[MPMediaQuery alloc] init];

NSLog(@"Logging items from a generic query...");
NSArray *itemsFromGenericQuery = [everything items];
for (MPMediaItem *song in itemsFromGenericQuery) {
    NSString *songTitle = [song valueForProperty: MPMediaItemPropertyTitle];
    NSLog (@"%@", songTitle);
}
```

Note: As with all code examples in this document, this example will run correctly only on a provisioned iPhone device, not in the Simulator.

To construct a more specific query, apply one or more media property predicates to a generic query. A predicate specifies a single logical condition to test media items against.

Listing 4-2 creates a predicate containing the condition that a media item's "artist" property must have a particular value. The listing then adds the predicate to a query.

Listing 4-2 Constructing and applying a media property predicate

```
MPMediaPropertyPredicate *artistNamePredicate =
    [MPMediaPropertyPredicate predicateWithValue: @"Happy the Clown"
                                     forProperty: MPMediaItemPropertyArtist];

MPMediaQuery *myArtistQuery = [[MPMediaQuery alloc] init];
[myArtistQuery addFilterPredicate: artistNamePredicate];

NSArray *itemsFromArtistQuery = [myArtistQuery items];
```

If you were to run this code, the `itemsFromArtistQuery` array would contain only those items from the iPod library that are by Happy the Clown.

You can construct and add multiple predicates to a query to narrow what the query matches. Listing 4-3 shows this technique using two predicates.

Listing 4-3 Applying multiple predicates to an existing media query

```
MPMediaPropertyPredicate *artistNamePredicate =
    [MPMediaPropertyPredicate predicateWithValue: @"Sad the Joker"
```

```
                                    forProperty: MPMediaItemPropertyArtist];

MPMediaPropertyPredicate *albumNamePredicate =
    [MPMediaPropertyPredicate predicateWithValue: @"Stair Tumbling"
                                     forProperty: MPMediaItemPropertyAlbumTitle];

MPMediaQuery *myComplexQuery = [[MPMediaQuery alloc] init];

[myComplexQuery addFilterPredicate: artistNamePredicate];
[myComplexQuery addFilterPredicate: albumNamePredicate];
```

You construct each predicate using a value of your choice (such as an artist's name) along with the appropriate media item property key. These keys are described in *MPMediaItem Class Reference* in `General Media Item Property Keys` and `Podcast Item Property Keys`.

When you apply more than one predicate to a query, the query combines them using the logical AND operator.

Important: Do not apply more than one predicate for a particular media item property. For example, do not specify two `MPMediaItemPropertyArtist` predicates to a query. If you do, the resulting behavior upon using the query is undefined.

You can also add predicates to a query upon initialization, as shown in Listing 4-4. (The two predicates in this example are assumed to be previously defined.)

Listing 4-4 Applying multiple predicates when initializing a media query

```
NSSet *predicates =
    [NSSet setWithObjects: artistNamePredicate, albumNamePredicate, nil];

MPMediaQuery *specificQuery =
    [[MPMediaQuery alloc] initWithFilterPredicates: predicates];
```

Listing 4-4 first defines an `NSSet` object that contains two predicates, then allocates and initializes a media query using the `initWithFilterPredicates:` class method.

Only certain property keys can be used to build valid predicates. These keys are tagged as "filterable" in *MPMediaItem Class Reference*. If you attempt to use a query that contains an invalid predicate, the resulting behavior is undefined.

Important: The system permits you to create predicates using property keys that are not meaningful for use with predicates. If you attempt to apply such a predicate to a query, the resulting behavior is undefined. To avoid problems with predicates:

- Read the *MPMediaItem Class Reference* documentation carefully.
- Code defensively by using the `canFilterByProperty:` method to validate media item property keys before using them.
- Don't let users define media property predicates.
- Thoroughly test your code to ensure that it behaves as you expect it to under various conditions.

Listing 4-5 shows how to use the `canFilterByProperty:` class method to validate a media item property key before using it in a predicate.

Listing 4-5 Testing if a property key can be used for a media property predicate

```
if ([MPMediaItem canFilterByProperty: MPMediaItemPropertyGenre]) {

    MPMediaPropertyPredicate *rockPredicate =
        [MPMediaPropertyPredicate predicateWithValue: @"Rock"
                                         forProperty: MPMediaItemPropertyGenre];

    [query addFilterPredicate: rockPredicate];
}
```

In Listing 4-5, only if the indicated media item property key (in this example, `MPMediaItemPropertyGenre`) can be used to construct a valid predicate will the body of the if statement execute. Apple recommends that you always perform a check like this before constructing a media property predicate.

Retrieving Media Item Collections

You can use also use a media query to retrieve sorted and arranged collections of media items. The arrangement you get depends on the value you set for the media query's `grouping` property.

Listing 4-6 shows how to retrieve all the songs by a particular artist, with those songs arranged into albums. The example logs the results to the Xcode debugger console.

Listing 4-6 Using grouping type to specify media item collections

```
MPMediaQuery *query = [[MPMediaQuery alloc] init];

[query addFilterPredicate: [MPMediaPropertyPredicate
                               predicateWithValue: @"Moribund the Squirrel"
                                      forProperty: MPMediaItemPropertyArtist]];
// Sets the grouping type for the media query
[query setGroupingType: MPMediaGroupingAlbum];

NSArray *albums = [query collections];
for (MPMediaItemCollection *album in albums) {
    MPMediaItem *representativeItem = [album representativeItem];
    NSString *artistName =
        [representativeItem valueForProperty: MPMediaItemPropertyArtist];
    NSString *albumName =
        [representativeItem valueForProperty: MPMediaItemPropertyAlbumTitle];
    NSLog (@"%@ by %@", albumName, artistName);

    NSArray *songs = [album items];
    for (MPMediaItem *song in songs) {
        NSString *songTitle =
            [song valueForProperty: MPMediaItemPropertyTitle];
        NSLog (@"\t\t%@", songTitle);
    }
}
```

You can see in Listing 4-6 that the value of the media query's `collections` property is an array of arrays. The outer for loop iterates over the albums performed the specified artist. The inner for loop iterates over the songs in the current album.

The `MPMediaQuery` class includes several convenience constructors for creating queries that are preconfigured with a grouping type. The following statement, for example, attaches the "albums" grouping type to a newly-allocated query:

```
MPMediaQuery *query = [MPMediaQuery albumsQuery];
```

For descriptions of all the convenience constructors, see *MPMediaQuery Class Reference*.

Using Media Item Artwork

One of the most useful and high-impact properties of a media item is its artwork. To display artwork, you use Interface Builder as well as Xcode. The steps are as follows:

1. Add a `UIImageView` object to your view layout in Interface Builder.
2. Add an IBOutlet instance variable to your view controller class to connect to the `UIImageView` object.
3. Retrieve the artwork from the media item that owns it (having previously retrieved the media item as described in this chapter).
4. Convert the artwork to a `UIImage` object, then assign it to your layout's `UIImageView` object.

Listing 4-7 shows how to do steps 3 and 4.

Listing 4-7 Displaying album artwork for a media item

```
MPMediaItemArtwork *artwork =
    [mediaItem valueForProperty: MPMediaItemPropertyArtwork];
UIImage *artworkImage =
    [artwork imageWithSize: albumImageView.bounds.size];

if (artworkImage) {
    albumImageView.image = artworkImage;
} else {
    albumImageView.image = [UIImage imageNamed: @"noArtwork.png"];
}
```

The `noArtwork.png` file used in the last line of Listing 4-7 is a fallback image that you add to your Xcode project, for use when a media item has no associated artwork.

Document Revision History

This table describes the changes to *iPod Library Access Programming Guide*.

Date	Notes
2010-05-21	Minor changes.
2009-11-18	New document that explains how to use the Media Player framework to find and play media items in iOS.

Document Revision History

Glossary

application music player An object that plays media items without affecting the state of the iPod application. See also iPod music player, music player.

default media library The object that provides library change notifications to your application.

filterable Describes a media item property key that can be used to build a valid search predicate.

filtering Describes obtaining a subset of media items from the iPod library by employing a search based on one or more media property predicate objects.

Genius Playlist A sequence of music media items that is created algorithmically based on music characteristics. See also on-the-go playlist, Smart Playlist.

iPod library The set of media items on a device that a user has synced from iTunes on the desktop.

iPod music player An object that plays media items by controlling the built-in iPod application, sharing its state. See also application music player, music player.

media item A single piece of media (such as one song) in the iPod library. A media item can be of various types including audio, audio book, and podcast.

media item artwork An object that represents a graphical image, such as album cover artwork, associated with a media item.

media item collection A set of related media items from the iPod library.

media item picker A specialized view controller that you employ to provide a graphical interface for selecting media items.

media property predicate A filter in a media query for retrieving a specified subset of media items from the iPod library.

media query An object that contains a description of what to retrieve from the iPod library and how those retrieved items should be arranged.

music player An object that plays media items. See also application music player, iPod music player.

on-the-go playlist A sequence of media items created by a user on a device, rather than on a notebook or desktop computer. See also Genius Playlist, Smart Playlist.

playback queue A list of media items for playback by a music player.

playback state For iPod library access, describes the playback state of a music player; one of playing, stopped, paused, fast forward, or rewind.

predicate For iPod library access, a specification of a single logical condition to test media items against when searching the iPod library.

repeat mode For iPod library access, indicates whether or not the current playback queue repeats or plays through just once.

shuffle mode For iPod library access, indicates whether or not the current playback queue plays in user-specified order or in a shuffled order.

Smart Playlist A sequence of media items created automatically according to user-specified criteria. See also Genius Playlist, on-the-go playlist.