Entourage SDK Implementation Guide for iOS

Product Version 1.5.0.1476

Published: 06-Dec-2013 13:31

Gracenote, Inc.
2000 Powell Street, Suite 1500
Emeryville, California
94608-1804
www.gracenote.com

Table of Contents

About the Entourage™ SDK

The Automatic Content Recognition (ACR) SDK is the Software Development Kit for Gracenote's Entourage Platform. With ACR, your app can continuously monitor PCM (Pulse-Code Modulation) audio and report metadata results retrieved from Gracenote via asynchronous callback methods. Using the SDK, you can quickly and easily build Entourage/ACR apps with access to Gracenote's global video database of rich metadata, images, and third-party links. Developers who have used Gracenote's MusicID, VideoID, or Mobile SDKs will find familiar API signatures and protocols.

PCM audio data is typically input from a microphone, but you can also use PCM audio data from any other source and submit it manually as byte data. If your app takes input from a microphone, it will need to link with the Apple AudioToolbox Framework.

The Entourage SDK supports iOS applications and includes:

- The Entourage iOS Objective-C API
- A Sample Application demonstrating Entourage running on the iOS platform
- Technical Documentation

Implementing an Entourage Client Application

Creating an Entourage client application involves the following general steps:

1. With your Gracenote provided license file, allocate and initialize an Entourage SDK Manager object (`GnSdkManager`).
2. With your SDK Manager object and Gracenote provided Client ID and Client Tag, allocate and initialize a User object (`GnUser`).
3. With your User object, allocate and initialize an ACR object (`GnACR`).
4. In your ACR object, set the two delegate methods that will handle Entourage/ACR status messages and metadata results.
5. Configure your ACR object for Audio input - allocate and initialize an Audio Config object (`GnAcrAudioConfig`)
6. Collect and submit PCM audio data.
7. Process ACR status messages and results via callback delegate methods. Results may be returned for a variety of reasons, from changes in the audio characteristics to the passing of time.
8. When done, stop Entourage.

Prerequisites

Before starting the Entourage service, you need three items from Gracenote Professional Services:

1. License File
2. Client ID
3. Client Tag

These three items are used to authenticate your app's access to Entourage/ACR. You should already have used these three items to build and run the Sample Application as detailed in the *Entourage SDK Getting Started Guide for iOS.* The authentication information is stored in your SDK manager object and is used in subsequent calls to Gracenote cloud-services.

If your application is implementing the MusicID module (see *Using MusicID*), you will need an additional Client ID, separate from the one used for ACR/Video/EPG. You must allocate separate User objects for the two Client IDs and use them with the appropriate queries (e.g., MusicID User object with MusicID queries). Your license needs to enable both Client IDs.

NOTE: All the sample code in the following sections is taken from the iOS Sample Application that comes with the SDK.

Local Lookup From Cache

Beginning with Entourage 1.2., your application has the option to store audio fingerprint data for local lookup, which can improve performance. In conjunction with Gracenote Professional Services, you can package fingerprints and metadata in entities known as *bundle*s, which can then be ingested and stored in local cache for Entourage SDK access. See *Storing Local Audio Fingerprint Data (Bundles)* for more information.

Local Lookup, Live TV, or Video Archive Recognition

The Gracenote Service provides content recognition from searching either local cache, a Video Archive, or live broadcast/pre-recorded TV data. The service can return one or more matches per request.

Recognition search order:

1. Local cache (if it exists) is searched first and can return either one or both of the following in one match:
    1. Zero or one Custom Data object.
    2. Zero or one AV Work object.
2. Online data is searched if there is no match from local cache. One online match can, currently, return one of the following but not both:
    1. One TV Airing object from a live or pre-recorded match.
    2. One AV Work object from a Video Archive match.

For an online search, which is searched first - Video Archive or live TV/pre-recorded data - depends on licensing.

For more information on Custom Data objects, see *Storing and Accessing Custom Data in Local Cache*.

You can use the ACR object's `lookupCacheOnly` property to restrict lookups to local cache (`true`) or allow online lookups (`false`). Default is `false`.

A TV Airing object can contain an identifier for fetching an associated AV Work object.

The following code sample shows how to get the three possible objects that can be returned in a match:

```
//** Get the enumerator to access the ACR Match objects (GnAcrMatch)
NSEnumerator *matches = result.acrMatches;

//** For each GnAcrMatch returned in GnResult
for (GnAcrMatch *match in matches) {

    //** Get the live TV match
    GnTvAiring *airing = match.tvAiring;

    //** Get the Video Archive match
   GnVideoWork *work = match.avWork;

   //** Get the Custom Data match
   GnCustomData *customData = match.customData;
}
```

Note About Error Handling

The wrapper code makes calls into the GN SDK and passes back any error messages unfiltered, which your code should handle, e.g.:

```
//**
//** Call the ACR object constructor
//**
 error = [mACR audioInitWithAudioConfig:config];

//**
//** Handle error
//**
if (error)
{

   NSString *msg = [NSString stringWithFormat:@"Audio init ERROR: %@ (0x%x)", error.localizedDescription,
error.code];

   //**
   //** Display error message to user in status display area
   //**
   [self updateStatusMessage:msg];
   return;
}
```

Refer to the GN SDK documentation for a list of error codes and descriptions.

Allocate and Initialize an SDK Manager Object

The first thing your app needs to do is allocate and initialize an SDK Manager object with the contents of the license file you obtained from Gracenote Professional Services. The SDK Manager object is used to monitor an app's interaction with Gracenote.

```
//** File: GN_ACR_SDKViewController.m
//** Method: acrButtonPressed

/*
 * Allocate and initialize a GnSdkManager instance with defined license file contents
 * The define below can be found in the Sample App's Resources/CustomerCredentials.h
 */
#define LICENSE_STRING    @"YOUR LICENSE FILE CONTENTS HERE"

NSError *error = nil;
GnSdkManager *mManager;
mManager = [[GnSdkManager alloc] initWithLicense:LICENSE_STRING error:&error];
```

Allocate and Initialize a User Object

Most devices will only have one user; however, on a server, for example, there could be a number of users running the app. Gracenote uses the ClientID to verify that the number of licensed and allowable users has not been exceeded.

```
//** File: GN_ACR_SDKViewController.m
//** Method: getUser
//**
//** Replace this define with your Gracenote-provided Client ID/Tag
//** The define below can be found in the Sample App's Resources/CustomerCredentials.h
//**

#define CLIENT_ID       @"YOUR CLIENT ID HERE"
#define CLIENT_ID_TAG      @"YOUR CLIENT TAG HERE"

//**
//** Note that the appVersion parameter can be set to any string value you like, however,
//** leaving it blank will generate an "invalid parameter" error. This field is legacy and is not
//** really used for anything anymore.
//**
mUser = [[GnUser alloc] initWithClientId:CLIENT_ID
                        clientIdTag:CLIENT_ID_TAG
                        appVersion:@"any string you prefer, e.g. '1.0'"
                        registrationType:GnUserRegistrationType_NewUser error:&error];
```

The `GnUser` object also has a constructor you can use with a user retrieved from serialized storage.

```
//** File: GN_ACR_SDKViewController.m
//** Method: getUser

//**
//** Retrieve user from serialized storage
//**
NSString *savedUser = [[NSUserDefaults standardUserDefaults] objectForKey:SAVED_USER_KEY];
user = [[[GnUser alloc] initWithSerializedUser:savedUser error:&error] autorelease];
```

After your app creates a new user, it should save it to serialized storage, where it can be retrieved every time your app needs it to allocate an ACR object.

If an app registers a new user on every use instead of storing a serialized user, then the user quota maintained for the Client ID is quickly exhausted. Once the quota is reached, attempting to create new users will fail. To maintain an accurate usage profile of your application, and to ensure that the services you are entitled to are not being used unintentionally, it is important that your application registers a new user only when needed, and then stores that user for future use.

Note that implementing MusicID requires its own Client ID and User object.

Allocate and Initialize an ACR Object

Once your app has allocated a User object, it can then be used to allocate an ACR object. This object is used to set the delegate methods for receiving and handling status messages and metadata results as well as submitting PCM audio data to Gracenote. There are a number configurable properties that can be set for this object (see *Other ACR Configurable Object Properties* below).

```
// File: GN_ACR_SDKViewController.m
// Method: acrButtonPressed

GnACR *mACR;
mACR = [[GnACR alloc] initWithUser:joe error:&error];
```

Configurable ACR Object Properties

The ACR object has a number of configurable user properties summarized in the table below. These parameters can affect network access and battery life.

Property	Usage Notes

`preferredMaxQueryInterval`	Type: `int` Default: 300 (seconds) Range: 0 - 86400 (1 day) Description: Sets the maximum number of seconds allowed before a recognition should be performed. The lowest setting to trigger queries is 1 second. Setting the interval to 0 means no forced queries from this timer (off). A low value (approximately 10 seconds or less) could mean that continuous querying will take place. This parameter exists in case transitions are not detected. For example, room background noise could mask the channel changes. In this case a recognition is not performed. If, however, this parameter is set to 180 seconds, for example, then after 180 seconds a query takes place. Any match rewinds this timer. Tradeoffs: A lower value forces more network traffic and CPU usage to generate a fingerprint when everything is working correctly and there are really no transitions to detect. It is battery intensive to keep doing unnecessary online queries. A higher value means it will take a long time to force a query and get the next match if transitions are not being detected. Values less than 60 require discussion with Gracenote Professional Services given the increased resource consumption. See *Local Cache and Online Query Logic (see page 21)* for more information.
`preferredMaxLocalQueryInterval`	Type: `int` Default: 0 ("off") Range: 0-86400 (1 day) Description: Sets the maximum number of seconds allowed before a recognition should be performed. The lowest setting to trigger queries is 1 second. Setting the interval to 0 means no forced queries from this timer (off). A low value (approximately 7 seconds or less) could mean that continuous querying will take place. This parameter exists to allow local lookups being executed more frequently independent from the `preferredMaxQueryInterval.` This is because lookups from the local cache are faster and less battery-consuming than online lookups. If a local query does not return a match, please note that **no followup online query is made**. See *Local Cache and Online Query Logic (see page 21)* for more information.
`noMatchModeTriggerLimit`	Type: `unsigned` Default: 60 Range: 0-9999999 Description: Configurable number of no matches before going into *no match* mode and increasing the back off timer value. Default is 60. This means that, after 60 no matches, the SDK will go into *no match* mode. Set to 0 to use default settings. The back off timer delays the time between queries. If you are getting a large number of no matches, then there is either silence or your app is capturing audio that cannot be identified.

`noMatchModeTimeIncrement`	Type: `unsigned` Default: 60 Range: 8-2592000 (30 days) Description: Configurable time increment (in seconds) to add to back off timer. Default is 60. This means that 60 seconds will be added to the time between lookups every time the `noMatchModeTriggerLimit` is reached.
`noMatchModeMaxTime`	Type: `unsigned` Default: 300 (5 minutes) Range: 60-5184000 (60 days) Description: Configurable maximum time (in seconds) for the back off timer when the app is in *no match* mode. Default value is 300 seconds (5 minutes). This means that once the back off timer reaches 300 seconds, it will no longer be incremented.

`optimizationMode`	Type: `NSString*`
	
	Default: `GnAcrOptimizationAccuracy`
	
	Description: Sets optimization mode (adaptive, accuracy , or speed) for ACR queries. Currently, the default is for accuracy. Optimizing for speed could result in slightly faster responses but could also increase the number of "no matches".
	
	Adaptive automatically optimizes match rate, precision, and battery consumption under current environmental conditions (e.g. audio level and quality, ambient noise, etc.). Generally, it works best in noisy environments and under most circumstances, but may not give you the speed or accuracy of the other modes. Support for this mode is limited. Please consult with Gracenote Professional Services before using this mode.
	
	One of four `const` string values:
	
	`GnAcrOptimizationDefault`
	`GnAcrOptimizationAccuracy`
	`GnAcrOptimizationSpeed GnAcrOptimizationAdaptive`

`lookupCacheOnly`	Type: `Bool` Default: `false` Description: • `true` - Restrict lookups to local cache. • `false` - Allow online lookups. Audio fingerprint data can be stored locally as well as online. This flag determines where lookups take place. See *Storing Local Audio Fingerprint Data (Bundles) (see page 18)* for information on creating locally stored data.

Parameters set outside their designated ranges are "snapped" to either the minimum or maximum allowed value. No error message is returned when this happens.

Configure the ACR Object for Audio Input

To initialize audio service, call the ACR object method `audioInitWithAudioConfig`, passing it a `GnAcrAudioConfig` configuration object. A suggested configuration object (`suggestedConfig`) is provided, but other configurations are possible; see "Audio Configuration Properties" below for more information.

```
//** File: GN_ACR_SDKViewController.m
//** Method: acrButtonPressed

/*
 *
 * Initialize audio with suggested configuration:
 *   Sample rate: 441000 (samples per second)
 *   Format : PCM16 - Specifies the amount of data used to represent each discrete amplitude sample.
 *                    16 bits (2 bytes), which gives a range of 65536 amplitude steps.
 *   Audio algorithm : Low quality (standard for microphone)
 *   1 channel (mono)
 */
GnAcrAudioConfig *config =
   [[[GnAcrAudioConfig alloc] initWithAudioSourceType:GnAcrAudioSourceMic
                               sampleRate:GnAcrAudioSampleRate44100
                               format:GnAcrAudioSampleFormatPCM16
                               numChannels:1] autorelease];
error = [mACR audioInitWithAudioConfig:config];
```

Audio Configuration Properties

The `GnAcrAudioConfig` object is set to an initial, "suggested" configuration (see code sample above). For most apps, the suggested configuration is optimal. However, you may want to modify this configuration for your unique environment or app.

Field	Type	Values	Default	Usage Notes
`algorithm`	`enum`	`GnAcrAudioSourceMic` `GnAcrAudioSourceLineIn`	`GnAcrAudioSourceMic`	For a microphone, use `GnAcrAudioSourceMic.` For higher quality input, such as a direct line-in from a TV, use `GnAcrAudioSourceLineIn.`
`sampleRate`	`enum`	`GnAcrAudioSampleRate8000` `GnAcrAudioSampleRate44100`	`GnAcrAudioSampleRate44100`	Indicates the sampling rate (http://en.wikipedia.org/wiki/Sampling_rate) per second.

`format`	`enum`	`GnAcrAudioSampleFormatPCM8`
		`GnAcrAudioSampleFormatPCM16`
			`GnAudioSampleFormatPCM16`	Specifies the format and amount of data used to represent each discrete amplitude sample.
				
				PCM8 = signed 8-bit integer linear PCM data.
				PCM16 = signed 16-bit integer linear PCM data. Little endian.

numChannels	NSUInteger	1 - Mono 2 - Stereo	1	

Code ACR Object Status and Result Delegate Methods

Your app needs to have an interface that implements `IGnAcrResultDelegate` and `IGnAcrStatusDelegate` and has an `acrStatusReady` method for handling statuses and an `acrStatusResults` method for handling metadata results.

```
//** File: GN_ACR_SDKViewController.m
//**
//** Sample App's main interface
//**
@interface GN_ACR_SDKViewController : UIViewController <IGnAcrResultDelegate, IGnAcrStatusDelegate> {
// ...


//** Method: acrButtonPressed

 mACR.statusDelegate = self;
 mACR.resultDelegate = self;

// ...

}
```

Your app needs to have an `acrStatusReady` method similar to the following for handling status messages:

```
//** File: GN_ACR_SDKViewController.m

-(void)acrStatusReady:(GnAcrStatus*)status
{
    //    self.status = status;

    NSString *message = nil;

    switch (status.statusType) {

       case GnAcrStatusTypeSILENT:
            message = [NSString stringWithFormat:@"Status: Audio Silent %10.2f", status.value];
            break;
       case GnAcrStatusTypeSILENT_RATIO:
            message = [NSString stringWithFormat:@"Status: Silent Ratio %10.3f", status.value];
            break;

        //**  ...

        default:
            break;
    }

    //**
    //** Display message in status display area
    //**
    [self updateStatusMessage:message];
}
```

Metadata Retrieval Workflow

In general, your delegate method for handling returned metadata results would implement some or all of the following process:

1. Iterate through matches with the `GnResult.acrMatches` parameter.
2. For each result, get deeper metadata (e.g., channel, program).
3. Possibly do subsequent Video Explore or EPG query (if you are licensed to do so) with result.

Your app needs to have an `acrResultReady` delegate method for handling metadata results similar to the following:

```
//** File: GN_ACR_SDKViewController.m

-(void)acrResultReady:(GnResult*)result
{
    @autoreleasepool {
        //** ACR query results will be returned in this callback
        //** Below is an example of how to access the result metadata.

        //** These callbacks may occur on threads other than the main thread.
        //** Be careful not to block these callbacks for long periods of time.


        //** Get the enumerator to access the ACR Match objects (GnAcrMatch)
        NSEnumerator *matches = result.acrMatches;
        int count = 0;

        //** For each GnAcrMatch returned in the GnResult
        for (GnAcrMatch *match in matches) {
            count++;

            if (count == 1) {
                //** Remember the first match for use with secondary query
                [mLatestAcrMatch release];
                mLatestAcrMatch = [match retain];
            }

            //** Get the title and subtitle from this GnAcrMatch
            NSString *acrTitle = match.title.display;
            NSString *acrSubtitle = match.subtitle.display;

            if (!acrSubtitle) {
                acrSubtitle = @"";
            }

            //** Retreive the GnTvAiring from the GnAcrMatch
            GnTvAiring *airing = match.tvAiring;

            //** Retreive the GnTvChannel from the GnTvAiring
            GnTvChannel *channel = airing.channel;

            //** Get the Channel callsign for display
            NSString *channelCallsign = channel.callsign;

            //** Get the position (ms from beginning of work/program) of the GnAcrMatch
            NSString* matchPosition = match.actualPosition;
            NSString* positionFormatted = @"";

            if (matchPosition != nil) {

                int seconds = [matchPosition intValue] / 1000;
                NSInteger remindMinute = seconds / 60;
                NSInteger remindHours = remindMinute / 60;
                NSInteger remindMinutes = seconds - (remindHours * 3600);
                NSInteger remindMinuteNew = remindMinutes / 60;
                NSInteger remindSecond = seconds - (remindMinuteNew * 60) - (remindHours * 3600);
                positionFormatted = [NSString
stringWithFormat:@"%02d:%02d:%02d",remindHours,remindMinuteNew,remindSecond];
            }

            if(match.customData){
                NSString *customDataID = match.customData.dataID;
                NSString *resultString = [NSString stringWithFormat:@"ACR: %@ %@ (Match #%d)",
customDataID, positionFormatted, count];
                [self updateResultMessage:resultString];

            } else {
                NSString *resultString = [NSString stringWithFormat:@"ACR: %@ %@ %@ %@ (Match #%d)",
acrTitle, acrSubtitle, channelCallsign, positionFormatted, count];
                [self updateResultMessage:resultString];
            }

            //** Here are more examples of how to get metadata from a result
            if (0) {
                NSLog(@"Match Title :       %@", match.title.display);
                NSLog(@"Match subtitle:     %@", match.subtitle.display);
                NSLog(@"Match actual pos:   %@", match.actualPosition);
                NSLog(@"Match adjusted pos: %@", match.adjustedPosition);

                GnVideoWork *work = match.avWork;
                if (work) {
                    NSLog(@"work tui:        %@", work.tui);
                    NSLog(@"work tag:        %@", work.tuiTag);
                    NSLog(@"work title:      %@", work.title.display);
                    NSLog(@"work is partial: %@", work.isPartial?@"YES":@"NO");
                }
                else
                    NSLog(@"No AV Work for this ACR match");

                GnTvAiring *airing = match.tvAiring;
                if (airing) {
                    NSLog(@"airing start:   %@", airing.dateStart);
                    NSLog(@"airing end:     %@", airing.dateEnd);


                    GnTvChannel *channel = airing.channel;
                    if (channel) {
                        NSLog(@"channel tui:      %@", channel.tui);
                        NSLog(@"channel tag:      %@", channel.tuiTag);
                        NSLog(@"channel name:     %@", channel.name);
                        NSLog(@"channel callsign: %@", channel.callsign);
                        NSLog(@"channel number:   %@", channel.number);
                    }
                    GnTvProgram *program = airing.tvProgram;
                    if (program) {
                        NSLog(@"program tui:      %@", program.tui);
                        NSLog(@"program tag:      %@", program.tuiTag);
                        NSLog(@"program title:    %@", program.title.display);
                        NSLog(@"program subtitle: %@", program.subtitle.display);
                        GnVideoWork *programWork = program.avWork;
                        if (programWork) {
                            NSLog(@"program work tui:      %@", programWork.tui);
                            NSLog(@"program work tag:      %@", programWork.tuiTag);
                            NSLog(@"program work title:    %@", programWork.title.display);
                        }
                    }
                }
            }


        }
        if (count == 0) {
            [self updateResultMessage:[NSString stringWithFormat:@"ACR: No match (%@)", [self
currentTime]]];
```

```
        }
    }
}
```

Managing the Microphone

The Entrourage SDK provides a class - `GNAudioSourceiOSMic` - for gathering PCM audio input from the device microphone. This class provides four methods for microphone management and listening:

- Start listening - `start` or `startMicrophone`
- Stop listening - `stop` or `stopMicrophone`

The newer `*Microphone` APIs are designed to give developers a simpler interface with improved recognition, particularly in difficult audio environments. With these APIs, your application is responsible for managing the Audio Session. This includes setting the Audio Session to the appropriate category for your application and activating/deactivating the Audio Session as necessary. You may want to use these APIs if, for example, your app needs to restart the audio while running in the background. For more information, see Apple's Audio Session reference documentation. See also our Tech Note — *Entourage Best Practices to Achieve All-Day Interactivity* — for more information on conserving battery life and supporting identification while running in the background.

The following steps illustrate how to use the `GNAudioSourceiOSMic` class :

1. Allocate and Initialize a `GNAudioConfig` object with these recommended settings:
    - Number of channels: mono (1) or stereo (2)
    - Sample rate: 44.1K (entered as 44100.0)
    - Bytes per sample: 16-bit mono(2) or 16-bit stereo (4)
2. Allocate a `GNAudioSourceMic` object using your `GnAcrAudioConfig` object.
3. Instantiate a class which implements `GNAudioSourceDelegate` and code an `audioBytesReady` method for it. The SDK calls this method when recorded audio is available.
4. Set the class that implements `GNAudioSourceDelegate` as a delegate to your `GNAudioSourceiOSMic` object.
5. To start listening, call `start` or `startMicrophone`
6. To stop listening, call `stop` or `stopMicrophone`

Using `GnAudioSourceiOSMic` code sample:

```
//** Class which implements GNAudioSourceDelegate
@interface
GN_ACR_SDKViewController : UIViewController <IGnAcrResultDelegate, IGnAcrStatusDelegate,
AVAudioSessionDelegate, GnAudioSourceDelegate>
{

// ...
 GnAudioSourceiOSMic *mSource;
// ...

//** Allocate and initialize Audio Configuration object
 GnAcrAudioConfig *config =
         [[[GnAcrAudioConfig alloc] initWithAudioSourceType:GnAcrAudioSourceMic
            sampleRate:GnAcrAudioSampleRate44100
            format:GnAcrAudioSampleFormatPCM16
            numChannels:1] autorelease];

//** Create the audio source object
mSource = [[GnAudioSourceiOSMic alloc] initWithAudioConfig:config];

//** Specify the delegate object which will receive the audio data callback
mSource.audioDelegate = self;

//** Start the audio Source
error = [mSource start];

//...

//**
//** Sample GnAudioSourceDelegate audioBytesReady method
//**
-(void)audioBytesReady:(void const * const)bytes length:(int)length
{
    //**
    //** The GnAudioSourceiOSMic object pointed to by mSource will call this method when audio is available
    //** from the input device. The application can then pass the audio into the GnAcr writeBytes method to
feed audio
    //** to ACR, or use the audio otherwise, by copying to memory, file, etc. This audio can also be fed
into
    //** GnMusicID writeBytes to do a music query.
    //** This callback is called from a high priority background thread. It is important to not block this
    //** callback for very long. It is advised to offload UI updates and expensive computations to the main
thread.
    //**
    NSError *error = nil;
    error = [mACR writeBytes:bytes length:length];
    if (error) {
        NSLog(@"audioBytesReady error: %@", error);
    }
}
```

Important

It is recommended that developers use the `GnAudioSourceMic` class if possible. Recognition will be improved through enhancements to this class' low-level processing without any application changes. Note that this class does not handle microphone interruptions or route changes. To learn more about handling these types of interruptions, familiarize yourself with Apple's `AVAudioSession` documentation:

http://developer.apple.com/library/ios/#documentation/Audio/Conceptual/AudioSessionProgrammingGuide/Introduction/Introduction.html (http://developer.apple.com/library/ios/#documentation%2FAudio%2FConceptual%2FAudioSessionProgrammingGuide%2FIntroduction%2FIntroduction.html)

Collect and Submit PCM Audio Data

The ACR object has a method - `writeBytes` - that allows you to submit PCM audio data collected from any source, including the device microphone. You need to pass it byte data of a specified length. How your app collects PCM data should align with your `GnAcrAudioConfig` object's configuration values, e.g., channels, sample rate and so on.

The Entourage SDK has a class - `GnAudioSourceiOSMic` - that should be used microphone for the best possible recognition. See the prior section - *Managing the Microphone*.

Using MusicID

Beginning with Entourage 1.2, MusicID is offered as a separate module that you can use in conjunction with ACR/Video/EPG. MusicID provides the following functionality:

- Stream Lookup - You can use PCM (Pulse Code Modulation) audio in an identification query (e.g., a "findAlbums" or "findTracks" query).
- Text Lookup - Search for album, track, artist, and so on, with a user-entered or application-generated string.
- Music Metadata - Based on query, objects are returned with metadata about tracks, albums, contributors, credits, artists, mood, tempo, and so on.

MusicID APIs are similar to the Video Explore and EPG APIs, but with added APIs for doing audio fingerprint searches. The MusicID APIs are implemented as *manual* synchronous queries, as opposed to the *automatic* ACR monitoring, which they are not coupled to. For more information on MusicID APIs, see the reference documentation for the `GNMusicID` class.

The ACR music classification status - `GnAcrStatusTypeMusic` - describes the previous 3 seconds of audio. As such, it can act as a trigger to do a music query, but the application needs to have already cached the previous audio. See the Sample Application for more information.

For other use cases than "on-demand" identification, consult with Gracenote Professional Services.

The configuration you use for gathering audio needs to be the same as that for fingprinting, e.g.;

```
//** Set the Audio config on the ACR object
 GnAcrAudioConfig *config =
          [[[GnAcrAudioConfig alloc] initWithAudioSourceType:GnAcrAudioSourceMic
                             sampleRate:GnAcrAudioSampleRate44100
                             format:GnAcrAudioSampleFormatPCM16
                             numChannels:1] autorelease];

//** Make sure these params are the same as you set up in the audio input
[musicQuery fingerprintBegin:GnMusicIdFingerprintTypeGNFPX
                sampleRate:44100
                sampleSizeBytes:2
                numberOfChannels:1
                error:&error];
```

Storing Local Audio Fingerprint Data (Bundles)

Beginning with Entourage 1.2., your application has the option to store audio fingerprint data in local cache that the Entourage SDK can access for faster local lookup. You can package fingerprints and metadata in entities known as *bundle*s. You can create these bundles (in conjunction with Gracenote Professional Services) and *ingest* them into your application with methods provided through the `GnFPCache` class. Prior to using this class, your application needs to have an interface that implements the `GnFPCacheSourceDelegate` protocol and has a `readBundleData` callback method:

```
-(BOOL)readBundleData:(void*)data
             capacity:(size_t)capacity
         numBytesRead:(size_t*)numBytesRead
{
    //** Load the requested number of bytes into the given data buffer.
    //** This callback will be called repeatedly until the ingestion is complete.

    NSData *fileData = [bundleFileHandle readDataOfLength:capacity];
    [fileData getBytes:data length:fileData.length];

    //** Report actual bytes read
    *numBytesRead = fileData.length;

    //** The return value indicates whether an error occured and
    //** that the ingestion should be aborted
    return NO;
}
```

This method is invoked when an application calls the `GnFPCache`'s `ingest` method. The delegate is given a data buffer and a capacity. Your application should write data into the buffer up to capacity bytes and update the `numBytesRead` value to reflect the actual number of bytes written to the data buffer. Updating `numBytesRead` to zero indicates there is no more data to ingest.The return value should indicate if an error occurred (file i/o, newtork failure, etc.). Returning YES indicates that an error occurred, at which point the ingestion is aborted.

This `GnFPCache` class has three methods:

1. `setStorageFolder`—Sets a local storage location for ingested bundle data. The system uses this location internally to cache bundle data to disk. This location is unrelated to the ingested source data.
2. `ingest`—Takes a stream source (either file or network) and calls the passed delegate's `readBundleData` method to read in data and store it locally.
3. `clearCache`—Clears all stored fingerprint bundle data.

To implement local audio fingerprint storage:

1. Work with Gracenote Professional services to create audio fingerprint bundle(s)
2. In your application, code an interface that implements the `GnFPCacheSourceDelegate` protocol and has a `readBundleData` callback method
3. In your application, call the `GnFPCache.setStorageFolder` method to set a local storage folder location

   If you do not set the storage folder, your application may crash during local lookup.

4. Call the `GnFPCache.ingest` method (passing it your `GnFPCacheSourceDelegate` object and stream source) to read in your audio fingerprint bundles and store them locally

You can use the ACR object's `lookupCacheOnly` property to restrict lookups to local cache (`true`) or allow online lookups (`false`). Default is `false`.

Ingesting fingerprint bundle code sample:

```
-(IBAction)loadFPBundle:(id)sender
{
    NSError *error = nil;
    [self initManager:&error];

    //** Set system storage location for FP Cache
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask, YES);
    NSString *cacheDirectryPath = [paths objectAtIndex:0];
    [GnFPCache setStorageFolder:cacheDirectryPath error:&error];

    //** This app looks in the Documents folder for a subfolder called "bundles" containing
    //** files to ingest. Then it loads each bundle in turn. Your app may load bundles
    //** as you see fit.
    paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
    NSString *docDirectryPath = [paths objectAtIndex:0];

    //** Find folder called bundles
    NSString *bundlesFolder = [docDirectryPath stringByAppendingString:@"/bundles/"];
    NSArray *directoryContent =
    [[NSFileManager defaultManager] contentsOfDirectoryAtPath:bundlesFolder error:&error];

    for (NSString *item in directoryContent)
    {
        //** Initiate the ingestion of each file in turn.
        NSString *fullPath = [bundlesFolder stringByAppendingString:item];
        bundleFileHandle = [NSFileHandle fileHandleForReadingAtPath:fullPath];

        //** The callback readBundleData:capacity:numBytesRead: will be called to
        //** load the data before ingest: will return.
        [GnFPCache ingest:self error:&error];

    }
}
```

Storing and Accessing Custom Data in Local Storage

When working with Gracenote Professional Services to create bundles for ingestion into local cache, you can also provide, for each fingerprint, custom data. When the Entourage SDK returns a match using local cache, you can access the `GnResult customData` object, which contains your custom data as bytes and an identifier. You are responsible for the identifier (during bundle creation), as well as the data's format and structure and how your application should process it.

Performing a Manual Lookup

To request a lookup immediately, without waiting for changes to be detected in the content, call the ACR object's `doManualLookup` method. This method queues a request for the next possible submit time.

⚠ Entourage must have been started to call this function.

Local Cache and Online Query Logic

A local-only cache query is made when:

- The ACR object's `lookupCacheOnly` field is `true`.
  The maximum time between local-only lookups (set with the ACR object's `preferredMaxLocalQueryInterval` field) is reached. Note that, in this case, if `lookupCacheOnly` is also `false`, the no-match counter does not get incremented.

A local-and-then-online lookup query is made when:

- A transition is detected
- A manual lookup is made
- The maximum time between online lookups (set with the ACR object's `preferredMaxQueryInterval` field) is reached
- `preferredMaxLocalQueryInterval` is 0 or not set

Note: If any of the first three conditions occurs during a fingerprint generation for a local-only lookup, the lookup is changed into a local-and-then-online lookup.

A local-only lookup cannot override a local-and-then-online lookup that is currently in progress.

Using Video Explore and EPG

While ACR gives you the capability to identify a live TV program, to get more in-depth information about a show (credits, contributors, external IDs, etc.) you will need to make Video Explore (http://www.gracenote.com/products/videoexplore/) (provides access to Gracenote's global database of rich video metadata, visual assets, trailers, and commerce links) or EPG (http://www.gracenote.com/products/eyeq/) (Electronic Program Guide) queries. Note that you will need additional licensing for this.

The iOS Sample App demonstrates how to make Video Explore and EPG queries (see portions below). See the *Entourage API Reference* for specific Video Explore and EPG API information.

Note that for the sake of brevity, most of the error checking and cleanup code has been removed from these examples.

Video Explore Example

```
//** File: GN_ACR_SDKViewController.m
//** Method: secondaryQueryBackgroundTask

//**
//** Get latest ACR match and associated AV work
//**
GnAcrMatch *theMatch = [[mLatestAcrMatch retain] autorelease];
GnVideoWork *theWork    = theMatch.avWork;

if (theWork)
{
    //**
    //** Try querying for work data
    //**
    GnVideo *videoQuery = [[GnVideo alloc] initWithUser:joe error:&error];
    GnResult *result = [videoQuery findWorksWithWork:theWork error:&error];

    NSEnumerator *works = result.videoWorks;

    //**
    //** Look thru works from Video Explore call. Display each work title
    //**
    for (GnTvProgram *work in works) {
        NSString *displayString = [NSString stringWithFormat:@"Secondary Query: %@", work.title.display];
        [self updateResultMessage:displayString];
    }
?}
```

Video Explore Caching

If you are using Video Explore, or other GN SDK components (Link, MusicID, etc.), there are options you can set to configure the local caching of metadata results and queries. Before going online, local cache, if it exists, is always searched first to improve response time. You can find these option properties in the `GnStorageSqlite` object. There are get and set accessors for each, e.g., `setStorageFolder`, `getStorageFolder`.

Property	Type	Usage Notes
`storageFolder`	`NSString*`	Storage file path.
`storageFileSize`	`NSUInteger`	Maximum storage file size in kilobytes, e.g., 1024 == 1MB. 5MB (5120) is recommended. When this limit is reached, oldest entries are removed first. There is a one-week storage limit; after this, queries and results begin to lose their relevance.
`storageMemorySize`	`NSUInteger`	Maximum size of memory for storage in kilobytes.

The Sample App demonstrates setting these options:

```
//** File: GN_ACR_SDKViewController.m
//** Method: acrButtonPressed

[GnStorageSqlite setStorageFolder:documentDirectoryPath error:&error];
[GnStorageSqlite setStorageFileSize:5120 error:&error];     //** 5 MB
[GnStorageSqlite setStorageMemorySize:512 error:&error];    //** 512 KB
```

EPG Example

```
//** File: GN_ACR_SDKViewController.m
//** Method: secondaryQueryBackgroundTask


//**
//** Get latest ACR match and associated TV program
//**
GnAcrMatch *theMatch = [[mLatestAcrMatch retain] autorelease];
GnTvProgram *theProgram = theMatch.tvAiring.tvProgram;

if (theProgram) {

    //**
    //** Do secondary query to get full program details such as credits and contributors
    //**
    GnEpg *epgQuery = [[GnEpg alloc] initWithUser:joe error:&error];
    GnResult *programResult = [epgQuery findProgramsWithProgram:theProgram error:&error];


    //**
    //** Look thru programs from EPG call. Display each program title
    //**
    NSEnumerator *programs = programResult.tvPrograms;
    for (GnTvProgram *program in programs) {
        NSString *displayString = [NSString stringWithFormat:@"Secondary Query: %@",
program.title.display];
        [self updateResultMessage:displayString];
    }

}
```

EPG Example - Retrieving External IDs

External IDs are 3rd party (Amazon, NetFlix, etc.) identifiers used to cross link this work's metadata with 3rd party services. They are not part of ACR core data, but you can retrieve them with an EPQ query, which requires additional licensing. Talk to Gracenote Professional Services about purchasing EPG licensing.

```
//** Example channel XID fetch

//**
//** Create an EPG query object with user object
//**
GnEpg *epgQuery = [[GnEpg alloc] initWithUser:joe error:&error];

//**
//** Acquire a TV channel object from GnAcrMatch/GnTvAiring
//** or synthesize one from tui/tag
//**
GnTvChannel *chan = [[GnTvChannel alloc] initWithID:@"251533198" tag:@"5F028079DB9DA1DF32F6D2914F5ADE7E"
idSource:GnIdSourceTypeTvChannel];

//**
//** Enable link data fetch
//**
epgQuery.enableLinkData = YES;

//**
//** Perform the query
//**
result = [epgQuery findChannelsWithChannel:chan error:&error];
[epgQuery release];

//**
//** Iterate through results and log external IDs
//**
NSEnumerator *channels = result.tvChannels;
for (GnTvChannel *channel in channels) {
     NSLog(@"Title:      %@", channel.name);
     NSLog(@"TUI:        %@", channel.tui);
     NSEnumerator *xids = channel.externalIDs;
     for (GnExternalID *xid in xids) {
            NSLog(@"%@", xid);
     }
}
```

Data Dictionary

An Entourage search result returns a variety of metadata fields that can be used to enrich the user experience. See *The Entourage SDK Data Dictionary* for a complete metadata results listing.

Entourage iOS APIs

For detailed information about the Entourage APIs, see the *Entourage (ACR) iOS API Reference* included with the SDK.