Mobile Client SDK for iOS Developer's Guide

Release Number 2.5.7.12

Published: 1/8/2013 6:29 PM

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

Getting Started with the iOS Sample Application

Introduction

The Mobile Client Software Development Kit (SDK) provides a Sample Application that demonstrates basic functionality. The SDK also provides a development project that is an example of how to incorporate the Mobile Client into your iPhone application.

This document describes how to integrate the Sample Application project into your development environment.

Preparing Your iPhone Development Environment

iOS development requires a specific environment. For details on setting up this environment, refer to the instructions on the Apple iOS Dev Center: http://developer.apple.com/devcenter/ios/

You also must register as an Apple Developer to download the software required to build and run the Gracenote iOS Mobile Client sample application. To register, go to http://-/developer.apple.com/programs/register/

Requirements

The following are required to build and run the sample application:

General Requirements

- Apple Developer Provisioning Certificate
- Apple App ID
- Apple Provisioning File

For Macintosh OS X Version 10.7 (Lion):

- Xcode Version 4.x
- iOS SDK Version 4.x

For Macintosh OS X Version 10.6 (Snow Leopard):

- Xcode Version 3.x and higher
- iOS SDK Version 4.x

Installing the iOS SDK

The Xcode project for the Mobile Client Sample Application uses the iOS v4.2

SDK. You must ensure that you have a compatible iOS SDK available in your development environment. The application is compatible with iOS v3.1.3 and later.

Follow the steps below to install the appropriate iOS SDK:

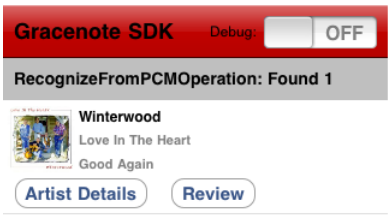
1. Go to the iOS Dev Center: http://developer.apple.com/devcenter/ios/index.action
2. Download the iOS SDK compatible for your development environment OS X version (see Technical Requirements). The download will also include the required version of Xcode.
3. Ensure Xcode and iPhone simulator are closed.
4. Run the downloaded iOS package (Xcode and iOS SDK.mpkg). This launches the iOS SDK installer. Once the installer has completed, tthe iOS SDK will be available as an Active SDK in Xcode.

Running the Sample Application

1. Unpack the Gracenote iOS Mobile Client Package: GN_Music_SDK_iOS.zip
2. Launch Xcode.
3. Close the Welcome to Xcode launch screen.
4. Choose File > Open.
5. Navigate to the parent directory of your unzipped package: GN_Music_SDK_iOS.
6. Beneath the parent directory, locate the file GN_Music_SDK_iOS.xcodeproj. Select this project and click Open at the bottom of the window. The Xcode project opens within Xcode.
7. Select the Overview drop down box at the top left corner of the application window.
8. In the first grouping, select Simulator.
9. Leave all other settings as is and click outside of the drop down box to exit it.
10. In the top middle of the window, click the Build and Run icon.
11. Look for Succeeded in the the bottom right of the window. If the build fails, see Mobile:Troubleshooting.
12. The Simulator will open and the Gracenote iOS Sample Application will launch.

13. Test the sample application by trying out some of the queries. When testing the application, note that:

    - The Recognize FPX and Recognize PCM queries use inputs underneath the GN_Music_SDK_iOS parent directory.
    - The Text Search and Lyric Fragment Search queries require you to manually input search terms.
    - The Recognize Mic query is unavailable with the iPhone Simulator at this time.



Troubleshooting

There are a few common reasons that the sample application build may fail:

Error Message: No architectures to compile for (ARCHS-i386 ppc, VALID_ARCHS=arm6 arm7)

Issue: Your development environment does not have the iOS SDK specified by the GN_Music_SDK_iOS project.

Potential Fix: The Gracenote iOS Sample Application project was created using iOS 4.2 SDK. If you have a newer version, change the Base SDK for the GN_Music_SDK_iOS target:

1. Click on the blue Info button at the top middle of the screen near Build and Run icon.
2. Select the Build tab, if it is not already selected.
3. In the Architectures section, select the Base SDK of the 4.x version that is installed on your system.

Error Message: Base SDK Missing in the Overview Dropdown.

Issue: This error normally occurs after a new SDK version and Xcode are installed. A base SDK needs to be designated.

Potential Fix: Follow the steps above to specify the Base SDK version.

Error Message: CodeSign error: code signing is required for product type 'Application' in SDK 'Device iOS x.x'

Issue: Xcode is attempting to deploy the application to an iPhone device unsuccessfully.

Potential Fix 1: If you are trying to use the simulator, change the project's Active SDK by selecting Simulator from the first grouping in the Overview drop down box in the top left hand corner.

Potential Fix 2: If you are trying to use a device, follow the steps listed in the Devices section of the iOS Provisioning Portal to ensure your device is correctly provisioned.

Mobile Client iOS Implementation Guide

2.5.7.12

Overview

This document is the Implementation Guide to the application programming interface (API) of the Gracenote Mobile Client software library. It provides conceptual background, implementation guidance, and example code to aid software developers in building application programs incorporating Gracenote Mobile Client services. For complete reference information on the Mobile Client API, see the included Reference Guide delivered as HTML pages.

Note

For conceptual simplicity, error checking has been omitted from most of the programming examples in this manual. In an actual production application, you will of course want to check the returned result code after each library call and terminate the logic flow or take appropriate recovery measures in case of failure. For more complete example code including full error checking, see the Sample Application included with the Mobile Client distribution package.

Deployment

Mobile Client is delivered as an Xcode project that can be integrated into an iPhone development environment that uses the Xcode integrated development environment (IDE).

Creating an iPhone Development Environment

iPhone development requires a specific environment. The details of creating such an environment can be obtained from http://developer.apple.com/iphone.

Before proceeding, ensure that your environment is equipped with the following:

- Macintosh OS
- Xcode
- iPhone SDK installed
- Apple Developer Provisioning Certificate
- App ID
- Provisioning File

To obtain the iPhone SDK, Apple Developer Provisioning Certificate, App ID, and Provisioning File, you must be a registered Apple Developer; see http://developer.apple.com/iphone for further details.

Using the Sample Application

The Mobile Client Sample Application is provided as source within an Xcode project. To run the Sample Application:

1. Unpack the Mobile Client distribution package.
2. Open the Xcode project within the package.
3. Add your client identifier to the Sample Application.
4. Build and run.

More detailed instructions are available in the Gracenote Mobile Client technical note Getting Started with the iOS Sample Application (included in this SDK).

Migrating to this Release

Deprecated, Renamed, and Changed Default GNConfig Parameters

Several GNConfig parameters were renamed in Mobile Client 2.5.2 to improve consistency and extensibility. Gracenote recommends you migrate your code to match the new naming conventions. However, this release supports the old parameter names to ensure backward compatibility.

The tables below summarizes the changes made, including the parameters that are currently supported but are deprecated. Deprecated parameters will be removed in a future release.

Deprecated Names

Old Name	New Name	Comments
country	content.country	As of Version 2.5.8., the default for this is null. Prior to 2.5.8, it was USA.

genre and genreId properties in GNSearchResponse	trackGenre and albumGenre	As of Version 2.5: Replace GNSearch-Response properties genre with trackGenre and genreId with albumGenre. Multiple genre levels can now be returned, so genres are delivered as a collection of GNDescriptor objects that contain a genre descriptor and a genre identifier. The properties genre and genreId will continue to return the descriptor and identifier from the lowest level genre returned to Mobile Client.
isGenreCoverArtEnabled	content.coverArt.genreCoverArt	
lang	content.lang	
preferredLinkSource	content.link.preferredSource	
web-services.coverArtSizePreference	content.coverArt.sizePreference	
webservices.gzipEnabled	N/A	

web-services.isInlineCoverArtEnabled	content.coverArt	As of Version 2.5. The default for this is false (as of version 2.5.8)
web-services.isSingleBestMatchPreferred	content.musicId.queryPreference.singleBestMatch	

Case Change Only

Cases are changed for consistency. Older case is supported. No deprecation.

Old Name	New Name
Content.contributor.images	content.contributor.images
Content.review	content.review
Content.contributor.biography	content.contributor.biography

Changed Default Values

Name	New Default Value	As of
content.coverArt	false	Version 2.5.8
content.country	null	Version 2.5.8 Prior to this, the default value was USA

Deprecated GNS

Deprecated GNSearchResponse Properties

Old Name	New Name
bestresponse.artistImage	bestresponse.contributorImage

Technical Requirements

Hardware	iPhone, iPad
Platform	iOS 3.1.3 or higher
Xcode version	3.2.3 or higher

Framework library size	1.1MB - 6.9MB [1]

[1] For details see Framework Size

Framework Size

The size of the framework reported above reflects the combined size of the universal framework for all three iOS architectures: armv6 and armv7 for the device, and i386 for the simulator. The total executable code for any single architecture is ~2.2MB. When linking the framework to your compiled application, the linker will automatically strip out the unneeded architectures as well as any framework symbols not utilized by your application. See the linker option "-dead_strip" for more information.

For example, the sample application GN_Music_SDK_iOS.app has a total executable size of 2.2MB for a combined armv6/armv7 architecture. This includes the app code and all linked SDK code. If the app is built for a single architecture, i.e. armv7 only, the compiled executable is 1.1MB. This is clearly much smaller than the entire SDK and illustrates how the linker will strip unused symbols, thus reducing the total size of the application.

See Xcode build settings for more information about setting build architectures and linker settings.

Apple Framework dependencies

Ensure that your environment is equipped with the following before proceeding:

- MediaPlayer.framework
- AudioToolbox.framework
- CoreData.framework
- CoreLocation
- AVFoundation.framework (must be weak-referenced in application)
- MapKit.framework
- CoreMedia
- libxml2.dylib

To enable file sharing, add Application supports iTunes file sharing = True in the plist file.

Xcode Build Settings

In your Xcode Build Settings, set Objective-C Automatic Reference Counting to No.

Linker Flags

The following linker flag must be used:

- -lsdtc++

Configuration and Authentication

Mobile Client uses a configuration object of type GNConfig to control its behavior. The behavior can be modified by altering the configuration object.

To obtain a configuration object, use the GNConfig:init method. This method returns a GNConfig instance that must be stored and used with Gracenote operations (see the section Operations). The method accepts a client identifier:

```
GNConfig* config;
// Generate configuration object
config = [GNConfig init: @"12345678-ABCDEFGHIJKLMNOPQRSTUVWXYZ012345"];
```

The client identifier is used to generate authentication information, which is stored in the object and used in turn to gain access to Gracenote's cloud-based services.

The configuration object can be customized by setting its properties via the GNConfig:setProperties method. For example, you can configure the preferred language for metadata returned from Gracenote:

```
// Set preferred language to Japanese
[config setProperty: @"content.lang" value: @"jpn"];
```

See the GNConfig.h Line 21 for a complete list of customizable properties.

Operations

An operation is a request performed by Mobile Client, such as creating a fingerprint or recognizing an audio stream. The Mobile Client class GNOperations provides methods for invoking operations.

Invoking Operations

Operations run asynchronously to the application invoking them, and return their results via a mechanism known as a result-ready object (described below under Receiving Results). Each operation must be provided a configuration object generated by GNConfig:init, along with a result-ready object to receive the results:

```
// Create result-ready object to receive recognition result
ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke operation
[GNOperations recognizeMIDStreamFromMic: searchResultReady config: config];
```

Operations call result-ready and status-changed methods in the application's main thread. As a best practice, any time-consuming or complex computational processes should not be run in the main thread. Doing so will block the UI and may cause the application to behave poorly and impact performance. For example, retrieving cover art should be run in a background thread.

Receiving Results

Result-ready objects implement one of the following Mobile Client protocols, depending on the type of operation:

- GNFingerprintResultReady
- GNSearchResultReady

Mobile Client calls the result-ready object's GNResultReady method when a result is generated. Your application can use this method to process the result:

```
// Result-ready object implements GNSearchResultReady protocol
@ interface ApplicationSearchResultReady : NSObject <GNSearchResultReady>
{
}
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Application code to process operation result
}
@ end
- (void) recognizeFromMic (GNConfig*) config
{
// Create result-ready object to receive recognition result
    ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke recognition operation
    [GNOperations recognizeMIDStreamFromMic: searchResultReady config: con-
fig];
}
```

Audio Recognition

Mobile Client provides three Gracenote technologies for audio recognition: MusicID-Stream, for audio delivered from a microphone or other streaming audio source (such as a radio signal or streaming Internet source), MusicID-File, for audio extracted from an audio file (such as a .wav or .mp3 file), and AlbumID, for identifying groups of audio files using fingerprints, text inputs, tag data, and Gracenote Identifiers. AlbumID can also use this data to group files into albums. The results of recognition operations are returned to the application via a result-ready object implementing the GNSearchResultReady interface.

If requested (and if the application is so entitled), Mobile Client can also provide cover art and Link identifiers with the recognition results; see Retrieving Cover Art and Retrieving Link Data, below, for further information.

Stream-based recognition can take longer than file-based. When recognizing an audio file, use file-based recognition to obtain the best response time.

MusicID-Stream

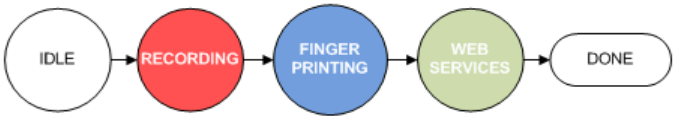
MusicID-Stream can be used to recognize a snippet of a song, such as a recording received from the device microphone or from an Internet stream.

Mobile Client provides two methods for invoking a MusicID-Steam recognition, one designed for simplicity, the other for flexibility.

GNOperations.recognizeMIDStreamFromMic

GNOperations.recognizeMIDStreamFromMic recognizes audio recorded from the microphone and is the simplest way to recognize music playing in the user's environment.

When invoked, Mobile Client obtains the device microphone and records 6.5 seconds of audio. This audio is processed and a MusicID-Stream fingerprint is generated. The fingerprint is then submitted to Gracenote Web Services for recognition. The result is delivered via an object that implements the GNSearchResultReady interface.



Status flow in stream-based audio recognition

The following example shows how to invoke GNOperations.recognizeMIDStreamFromMic.

```
// Create result-ready object to receive recognition result
ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke recognition operation
[GNOperations recognizeMIDStreamFromMic: searchResultReady config: config];
```

During audio recognition, Mobile Client sends status updates to notify the application of progress.

GNOperations.recognizeMIDStreamFromPcm

GNOperations.recognizeMIDStreamFromPcm recognizes audio provided as a buffer of PCM (pulse-code modulation) data. This provides the application with additional flexibility: for instance, the application can recognize audio from external streaming audio sources such as a radio signal or an Internet stream.

When invoked, Mobile Client reads the PCM audio data. The audio is processed and a MusicID-Stream fingerprint is generated. The fingerprint is then submitted to Gracenote Web Services for recognition. The result is delivered via an object that implements the GNSearchResultReady interface.

Status flow in PCM-based audio recognition

The following example shows how to invoke GNOperations.recognizeMIDStreamFromPcm.

```
// Create PCM sample buffer
GNSampleBuffer* sampleBuffer = [GNSampleBuffer gNSampleBuffer: samples
bytesPersample: 1 numChannels: 1 sampleRate: 8000];
// Create result-ready object to receive recognition result
ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke recognition operation
[GNOperations recognizeMIDStreamFromPcm: searchResultReady config: config
sampleBuffer: sampleBuffer];
```

During audio recognition, Mobile Client sends status updates to notify the application of progress.

The PCM audio sample must be at least 6.5 seconds long for Mobile Client to successfully generate a MusicID-Stream fingerprint.

Audio Sessions and Audio Categories

Mobile Client does not create an iOS audio session, and consequently does not set an iOS audio category when recording audio from the microphone. This leaves your application free to set the audio category appropriately. Although Mobile Client may still be able to record from the microphone even without setting the audio category, it is recommended that you do create an audio session and set the audio category yourself; this will allow your application to react appropriately to system events and state changes, such as when a phone call is received. For information on audio sessions and audio categories, see the Audio Session Programming Guide, available from Apple.

If your application needs to incorporate audio features beyond Mobile Client's microphone-recognition capability, it must create an audio session and set the audio category: for example, if the application is required to play audio and also use Mobile Client to recognize audio from the microphone, it must use the playing and recording audio category.

Recording audio categories should be used only while the device is actually recording. When using Mobile Client's recognize-from-microphone feature, set the audio category to the appropriate recording category before invoking the operation, then monitor the operation's status-changed events and switch to a non-recording category when recording has completed. See Status Change Updates, below, for more information.

For some applications, it may be inconvenient for Mobile Client to access the audio subsystem. In this case, audio can still be recognized through Mobile Client's recognize-from-PCM functionality. Your application can interact with the audio subsystem to obtain raw PCM audio data and provide it to Mobile Client for recognition.

MusicID-File

MusicID-File can be used to recognize an audio file.

Mobile Client provides two methods for invoking a MusicID-File recognition, one designed for simplicity, the other for flexibility.

GNOperations.recognizeMIDFileFromFile

GNOperations.recognizeMIDFileFromFile recognizes and audio file stored on the device.

When invoked Mobile Client decodes the audio file. The audio is processed and a MusicID-File fingerprint is generated. The fingerprint is then submitted to Gracenote Web Services for recognition. The result is delivered via an object that implements the GNSearchResultReady protocol.

Status flow in file-based audio recognition

Mobile Client can recognize audio files in the following formats:

- .wav
- .mp3
- .aac
- .caf
- .aaif

The following sampling rates are supported, in both monaural and stereo:

- 8000 Hz
- 11025 Hz
- 16000 Hz
- 22050 Hz
- 32000 Hz
- 48000 Hz

This functionality is available only for certain devices running iOS 4.0 or higher. As shown in the following table, all supported devices can recognize audio files stored in the document directory and most can also recognize files in the iPod library:

Device	Document directory	iPod library
iPod Touch, 4th generation	Yes	Yes
iPhone 3G	Yes	No
iPhone 3Gs	Yes	Yes
iPhone 4 or higher	Yes	Yes

Files containing video components are not supported.

To recognize audio files stored in the device's document directory, you must enable iTunes file sharing by setting Application supports iTunes file sharing = True in your application's plist file.

Also, iOS may occasionally deny access to files in the document directory based on the state and actions of other applications running on the device. This may happen if

- there is an incoming phone call
- your application is in the background and another application starts playback

In these cases, Mobile Client will return an appropriate error. It is recommended that you test your application to ensure that it responds appropriately to such iOS limitations.

Processing an audio file requires approximately 20 seconds of audio and that audio must come from the start of the track.

Invoking file-based recognition on an audio file stored in the iPod library will cause it to stop being played if it is currently being played by the iPod.

To recognize audio from a file, use the method GNOperations:recognizeMIDFileFromFile. This method requires a URL to the desired file, as shown in the following code fragment:

```
// Create URL from file path
NSURL* fileURL = [NSURL fileURLWithPath: filePath];
// Use URL to recognize audio file
RecognizeFromFileOperation* op = [RecognizeFromFileOperation rec-
ognizeFromFileOperation: config];
[GNOperations recognizeMIDFileFromFile: op config: config fileUrl:
fileURL];
```

GNOperations.recognizeMIDFileFromPcm

GNOperations.recognizeMIDFileFromPcm recognizes audio provided as a buffer of PCM (pulse-code modulation) data. This provides the application with additional flexibility: for instance, file formats not directly supported by Mobile Client can be decoded by the application to PCM and recognized via this method.

When invoked, Mobile Client reads the PCM audio data. The audio is processed and a MusicID-File fingerprint is generated. The fingerprint is then submitted to Gracenote Web Services for recognition. The result is delivered via an object that implements the GNSearchResultReady interface.

Status flow in PCM-based audio recognition

The following example shows how to invoke GNOperations.recognizeMIDFileFromPcm.

```
// Create PCM sample buffer
GNSampleBuffer* sampleBuffer = [GNSampleBuffer gNSampleBuffer: samples
bytesPersample: 1 numChannels: 1 sampleRate: 8000];
// Create result-ready object to receive recognition result
ApplicationSearchResultReady* searchResultready = [Appli-
cationSearchResultReady alloc];
// Invoke recognition operation
[GNOperations recognizeMIDFileFromPcm: searchResultready config: config sam-
pleBuffer: sampleBuffer];
```

During audio recognition, Mobile Client sends status updates to notify the application of progress.

Processing an audio file requires approximately 20 seconds of audio and that audio must come from the start of the track.

AlbumID

AlbumID is a powerful and flexible recognition processing tool that can be used to provide advanced recognition of audio files within the user's collection. By leveraging contextual information about the audio files, AlbumID can effectively identify, group and organize a collection, providing clean and consistent metadata. It is best used for:

- Analyzing groups of media files, where the grouping of results is as important as the accuracy of the individual results
- Receiving responses that match the contextual data of an audio file, such as metadata from ID3 tags

AlbumID can be used to recognize items in the application's document directory or in the device's iPod library. It uses a variety of combinations of the following recognition methods and inputs:

- MusicID-File fingerprinting: Audio files in supported formats are decoded and a MusicID-File fingerprint is then generated.
- Text search and text comparison: Metadata from ID3 tags extracted from supported file formats or additional text information provided by the application are used to search for appropriate tracks and albums.

- Gracenote Identifiers: Audio files sometimes have an associated Gracenote Identifier, which Mobile Client can use for consideration during the identification process.
- Audio file groupings: Files can be analyzed in groups, which allows common albums to be determined based on the tracks in the group.
- Audio file name and file-system (folder) location: As music collections are often grouped by directory (Artist/Album/Track), file name and location can also used during identification. Only files in the application's Document directory can be recognized.

Fingerprints cannot be generated for files containing video components.

To recognize audio files stored in the device's document directory, you must enable iTunes file sharing by setting Application supports iTunes file sharing = True in your application's plist file.

iOS may occasionally deny access to files in the document directory based on the state and actions of other applications running on the device. This may happen in the following cases:

- There is an incoming phone call
- Your application is in the background and another application starts playback

In these cases, Mobile Client will return an appropriate error. It is recommended that you test your application to ensure that it responds appropriately to such iOS limitations.

An audio file requires approximately 20 seconds of audio to successfully generate a fingerprint.

Invoking AlbumID with fingerprinting on an audio file stored in the iPod library will cause it to stop being played if it is currently being played by the iPod. The AlbumID configuration can be altered to omit fingerprinting

While commerce identifiers can be requested, AlbumID does not support the preference of a specific identifier over the actual Album a song came from. These preferred results can instead be obtained by using GNOperations.recognizeMIDFileFromFile or GNOperations.recognizeMIDFileFromPcm. See MusicID-File for more information.

Mobile Client provides various ways to invoke AlbumID, allowing the developer to choose between a simplified or more flexible implementation.

You can improve retrieval performance for AlbumID by retrieving enriched content in the background. For more information, see Improving Retrieval Performance by Using Enriched Content URLs.

Calling AlbumID Operations Serially

An application should call AlbumID operations serially (one at a time). An application should only call another operation after the previous operation has completed. If multiple AlbumID operations are called concurrently with many tracks (for example, 1000 tracks per AlbumID directory operation), it might impact performance on the device and might cause the application to run out of memory.

GNOperations.albumIdFromMPMediaItemCollection:config:collection

GNOperations.albumIdFromMPMediaItemCollection:config:collection can be used to recognize items from the devices iPod library. A collection of MPMediaItem objects can be provided for recognition.

The following code sample shows how to invoke GNOperations.albumIdFromMPMediaItemCollection:config:collection

```
// Using MPMediaPickerController get MPMediaItemCollection for selected
files from iPodLibrary
// Create result-ready object to receive recognition result.
// ApplicationSearchResultReady must implement the GNSearchResultReady
interface
- (void)mediaPicker: (MPMediaPickerController *)mediaPicker did-
PickMediaItems:(MPMediaItemCollection *)mediaItemCollection {
ApplicationSearchResultReady* searchResultready = [Appli-
cationSearchResultReady alloc];
// Invoke AlbumID operation with the result-ready object, a GNConfig object
// instance and a MPMediaItemCollection which contains iPod library files
to identify
[GNOperations albumIdFromMPMediaItemCollection:searchResultready con-
fig:config collection:mediaItemCollection];
}
```

GNOperations.albumIdDirectory:config:directoryPath

GNOperations.albumIdDirectory:config:directoryPath takes a single directory path and identifies all of the audio files in the directory tree, including sub-directories. This method can only recognize files in the application's Document directory.

The following code example shows how to invoke GNOperations.albumIdDirectory.

```
// Create result-ready object to receive recognition result.
// ApplicationSearchResultReady must implement the GNSearchResultReady
interface
ApplicationSearchResultReady* searchResultready = [Appli-
cationSearchResultReady alloc];
// Invoke AlbumID operation with the result-ready object, a GNConfig object
// instance and the root of the directory tree to be processed
[GNOperations albumIdDirectory:searchResultReady config:config direc-
toryPath:documentDirectryPath];
```

When invoked this method performs the following operations:

1. Searches the directory tree and locates audio files that are supported by Gracenote MusicID-File audio decoder or AlbumID tag decoder

2. Generates MusicID-File fingerprints for files in supported formats

3. Extracts information tags from supported formats which can include artist, album and track information and Gracenote Identifiers

4. The recognition inputs collected by the above steps are combined with the file name and path and delivered to the Gracenote Service for identification; this may result in multiple queries to the Gracenote Service

5. The results of identification are delivered to the application

The behavior of GNOperations.albumIdDirectory:config:directoryPath can be controlled via the AlbumID configuration parameters. See AlbumID Configuration for more information.

GNOperations.albumIdFile:config:filePaths

GNOperations.albumIdFile:config:filePaths takes the filenames and paths of a collection of audio files and applies AlbumID identification for grouping and organizing.

This method allows targeted identification of groups of audio files, or a single audio file, utilizing all of the recognition technologies available to AlbumID. This method can only recognize files in the application's Document directory.

The following code example shows how to invoke GNOperations.albumIdFile:config:filePaths.

```
// Create result-ready object to receive recognition result.
// ApplicationSearchResultReady must implement the GNSearchResultReady
interface
ApplicationSearchResultReady* searchResultready = [Appli-
cationSearchResultReady alloc];
// Assemble a collection of audio files filename and path
NSArray *filesToIdentify = [NSArray arrayWithObjects:filePath1,filePath2,
filePath3,nil];
// Invoke AlbumID operation with the result-ready object, a GNConfig object
// instance and a collection of files to identify
[GNOperations albumIdFile:searchResultReady config:config file-
Paths:filesToIdentify];
```

When invoked this method performs the following operations:

1. Generates MusicID-File fingerprints for files in supported formats

2. Extracts information tags from supported formats which can include artist, album and track information and Gracenote Identifiers

3. The recognition inputs collected by the above steps are combined with the file name and path and delivered to the Gracenote Service for identification; this may result in multiple queries to the Gracenote Service

The behavior of GNOperations.albumIdFile:config:filePaths can be controlled via the AlbumID configuration parameters. See AlbumID Configuration for more information.

GNOperations.albumIdList:config:list

GNOperations.albumIdList:config:list takes a collection of objects where each object contains the recognition inputs for an audio file. Recognition inputs are:

- MusicID-File Fingerprint
- Textual metadata:
    - Artist Name
    - Album Title
    - Track Title
    - Track Number
- File name and path
- Gracenote Identifiers

AlbumID does not require all of the above inputs to be effective; it can use only those provided to assist identification. Note this also means that text can be used if no fingerprint can be created, allowing AlbumID to be used for audio files that cannot be accessed or decoded by Mobile Client. Mobile Client does not provide a method to export Gracenote Identifiers from a file directly to the application; however, the query result contains these identifiers, which can be used for subsequent AlbumID List queries.

The following code example shows how to invoke GNOperations.albumIdList:config:list.

```
// Create result-ready object to receive recognition result.
// ApplicationSearchResultReady must implement the GNSearchResultReady
interface
ApplicationSearchResultReady* searchResultready = [Appli-
cationSearchResultReady alloc];
// Assemble a collection of GNAlbumIdAttributes objects.
// Each GNAlbumIdAttributes has the recognition inputs for a specific file.
ArrayList<GNAlbumIdAttributes> filesToIdentify = new Array-
List<GNAlbumIdAttributes>();
// Assemble the recognition inputs for individual audio files into GNAl-
bumIdAttributes instances
    GNAlbumIdAttributes * fileAttrib = [GNAlbumIdAttributes gNAl-
bumIdAttributes:@""
        identifier:@"List1"
        albumTitle:@"Keep the Faith"
        trackTitle:@""
        trackNumber:@""
```

```
        genre:nil
        artist:@"Bon Jovi"
        gnId:nil
        fingerPrintData:fingerprintString];
NSArray *filesToIdentify = [NSArray arrayWithObjects: fileAttrib,nil];
// Invoke AlbumID operation with the result-ready object, a GNConfig object
// instance and a collection of files to identify
[GNOperations albumIdList: searchResultready config:config list: files-
ToIdentify];
```

When this method is invoked, the recognition inputs for each file are delivered to Gracenote Web Services for identification, which may result in multiple queries to Gracenote Web Services.

The behavior of GNOperations.albumIdList can be controlled via the AlbumID configuration parameters. See AlbumID Configuration for more information.

AlbumID Configuration

The behavior of AlbumID can be controlled by appropriately configuring the GNCOnfig object provided when an AlbumID operation is invoked.

The configuration parameters are described below.

Parameter	Description	Default
content.albumId.queryPreference.useTagData	When true, AlbumID will use textual metadata when identifying an audio file.	true
content.albumId.queryPreference.useFingerprint	When true, AlbumID will use MusicID-Fingerprints when identifying an audio file.	true
content.albumId.queryPreference.useGN_ID	When true, AlbumID will use a Gracenote Identifier when identifying an audio file.	true

AlbumID Query Load

AlbumID is capable of recognizing large music collections and in most cases will use multiple queries for recognition of tracks and retrieval of related content.

To assist with recognition, Mobile Client will group audio files. The number of queries used for recognition is impacted by the number of groupings. Audio file groupings are based on:

- The metadata of the audio files (tag data is used to group similar tracks)
- The organization of the audio files (files can be grouped based on the directories they reside in)

- The limit of the recognition interface

Because file groupings are impacted by track metadata and organization, it is difficult to predict how many recognition queries will be needed.

Once the audio files have been recognized, your application can access related content via the result objects. Although queries are batched in multiple groups, the responses for all the grouped queries are provided to the application at one time, at the end of the operation.

The example below provides an indication of the number of queries that are required for AlbumID. This example can be extrapolated to larger collections requiring similar content to be delivered.

AlbumID Query Load Example

Consider an audio collection organized as shown below.

Assume:

- All tracks have accurate Artist Name, Album Title, and Track Title tag data.
- No Gracenote Identifiers are known for any of the files:
    - /sdcard/Red Hot Chili Peppers/By The Way/By The Way.mp3
    - /sdcard/Red Hot Chili Peppers/By The Way/Universally Speaking.mp3
    - /sdcard/Red Hot Chili Peppers/By The Way/This Is The Place.mp3
    - /sdcard/Red Hot Chili Peppers/By The Way/Dosed.mp3
    - /sdcard/Red Hot Chili Peppers/By The Way/Don't Forget Me.mp3
    - /sdcard/Red Hot Chili Peppers/By The Way/...
    - /sdcard/Midnight Oil/10, 9, 8, 7, 6, 5, 4, 3, 2, 1/Outside World.mp3
    - /sdcard/Midnight Oil/10, 9, 8, 7, 6, 5, 4, 3, 2, 1/Only The Strong.mp3
    - /sdcard/Midnight Oil/10, 9, 8, 7, 6, 5, 4, 3, 2, 1/Short Memory.mp3
    - /sdcard/Midnight Oil/10, 9, 8, 7, 6, 5, 4, 3, 2, 1/Read About It.mp3
    - /sdcard/Midnight Oil/10, 9, 8, 7, 6, 5, 4, 3, 2, 1/Scream In Blue.mp3
    - /sdcard/Midnight Oil/10, 9, 8, 7, 6, 5, 4, 3, 2, 1/...

Using albumIdDirectory, the base directory /sdcard would be provided. The AlbumID algorithm will create two groups, one for Red Hot Chili Peppers and one for Midnight Oil. An identification query is generated for each group.

After identification is completed the results are provided via a result-ready object. The application can then process the individual track results by processing the respective GNSearchResponse object.

Related content, such as Cover Art, is not included in the initial result set. When the application calls GNSearchResponse.getCoverArt(), a query is generated to fetch the cover art.

The application should only call GNSearchResponse.getCoverArt() for one track on each album. This is to avoid the same Cover Art from being fetched more than once. The application should include some intelligence to ensure this.

In this example the minimum number of queries required to identify the audio tracks and retrieve the track and album metadata and cover art for each album is four:

- Two queries for identification
- One query to retrieve Cover Art for Red Hot Chili Peppers' album By The Way
- One query to retrieve Cover Art for Midnight Oil's album 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Applications can minimize the number of queries made through correct configuration, and analysis and storage of results.

Single Best Match and Multiple Matches

The audio recognition operations MusicID-Stream and MusicID-File can return a single best match or multiple matches. The default configuration returns a single best match, but this can be changed by configuring the GNConfig instance provided when the recognition operation is invoked. The multiple match configuration parameter is described below.

Parameter	Description	Default
content.musicId.queryPreference.singleBestMatch	When true, single best match is returned for MusicID-Stream and MusicID-File recognition operations. When false, multiple matches are returned.	true

When configured for a single best match, the best match is identified by comparing the characteristics of all possible matches against criteria specific to your application. By setting properties in the configuration object, you can change the criteria used in determining the best match.

When multiple matches are configured, all matches are delivered, up to a maximum of ten. The application can apply its own criteria to determine the best match and deliver it to the user. The application can also allow the user to choose the best match.

Text and Lyric Fragment Search operations always return multiple matches and AlbumID always returns a single match.

Single best match queries can be configured to return cover art in the recognition response. Cover art is not delivered with the responses containing multiple matches; however, it can be retrieved with an additional query by Gracenote Identifier (see Retrieving Related Content by Gracenote Identifier).

Fingerprints

Gracenote Web Services uses audio fingerprints to match audio with metadata and cover art. Mobile Client can generate audio fingerprints from either of the following sources:

- Audio captured from the microphone or other streaming audio source
- Pulse-code modulation (PCM-encoded) audio stored in a buffer

Generating a Fingerprint

Fingerprints are returned via a result-ready object implementing the Mobile Client protocol GNFingerprintResultReady. Mobile Client calls the result-ready object's GNResultReady method when a result is generated. Your application can use this method to process the result:

```
// Result-ready object implements GNFingerprintResultReady protocol
@ interface ApplicationFingerprintResultReady : NSObject <GNFin-
gerprintResultReady>
{
}
@ end
// Provide implementation for GNResultReady method
@ implementation ApplicationFingerprintResultReady
- (void) GNResultReady: (GNFingerprintResult*) result
{
// Application code to process fingerprint result
}
@ end
```

The application must create the result-ready object and provide it when invoking the fingerprint generation operation:

```
// Create result-ready object to receive fingerprint result
ApplicationFingerprintResultReady* fingerprintResultReady = [Appli-
cationFingerprintResultReady alloc];
// Invoke fingerprint generation operation
[GNOperations fingerprintMIDStreamFromPcm: fingerprintResultReady config:
config sampleBuffer: sampleBuffer];
```

Searching by Fingerprint

To submit a fingerprint to Gracenote Web Services for matching, use the method GNOperations:searchByFingerprint:

```
// Create result-ready object to receive matching result
ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke matching operation
[GNOperations searchByFingerprint: searchResultReady config: config fin-
gerprintData: fp];
```

A collection of possible matches is returned as the search result. During fingerprint matching, Mobile Client sends status updates to notify the application as each of the stages shown in the figure is reached:

Status flow in fingerprint matching

Text Search

Using the method GNOperations:searchByText:config:artist:albumTitle:trackTitle, Mobile Client can perform a text-based search in the Gracenote database for an album title, track title, and/or artist name. A combination of any of the three fields can be provided to narrow the search results:

```
// Create result-ready object to receive search result
ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke search operation
[GNOperations searchByText: searchResultReady config: config artistName:
artistName albumTitle: albumTitle trackTitle: trackTitle];
```

A collection of possible matches is returned as the search result; to get an exact match, use a Gracenote unique identifier instead of a text search (see Retrieving Related Content by Gracenote Identifier, below). During the search process, Mobile Client sends status updates to notify the application as each of the stages shown in the figure is reached:

Status flow in text search

Lyric Fragment Search

Using the method GNOperations:searchByLyricFragment:config:lyricFragement:artist, Mobile Client can search the Gracenote database using a lyric fragment, optionally augmented by an artist name:

```
// Create result-ready object to receive search result
ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
```

```
// Invoke search operation
[GNOperations searchByLyricFragment: searchResultReady config: config lyr-
icFragment: lyricFragment artist: artistName];
```

A collection of possible matches is returned as the search result. During the search process, Mobile Client sends status updates to notify the application as each of the stages shown in the figure is reached:

Status flow in lyric fragment search

Retrieving Related Content

Mobile Client can retrieve content related to an audio recognition result, including:

- Genre
- Mood
- Tempo
- Origin
- Era
- Artist Type
- Cover Art
- Artist Images
- Artist Biographies
- Album Reviews

Related content can be retrieved from the 2.5.9.8 returned by an audio recognition operation or via a Gracenote unique album or track identifier.

For more information on using images returned by Mobile Client see Image Resizing Best Practice.

Genre

Mobile Client returns album and track level genre descriptors with a recognition or search result. Genre descriptors can be displayed to the end-user or can be used to categorize music in the user's collection

for organization, navigation, or playlist generation.

Genre Levels

Gracenote has multiple levels of genre descriptors. Each level describes the related music with a different amount of detail and granularity.

Two options for genre are available: DEFAULT and EXTENDED. The DEFAULT option returns a single, default genre descriptor in the result. The EXTENDED option returns multiple levels of genre descriptors in the result.

The recommended option to use depends upon the application, its use-cases, and the target audience. Some applications might want to use the DEFAULT option, as it is the simplest and contains the most commonly known genres. Other applications might want to provide a richer genre experience via the multiple levels in the EXTENDED option. Applications might also use the EXTENDED option to give their users a choice of which genre level is used.

A Mobile Client operation can be configured for one of these options via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.genre.level	Sets the option for the genre descriptors returned; DEFAULT and EXTENDED	DEFAULT

Genre Localization

Gracenote provides a global solution and Mobile Client can be configured to deliver genres specific to a supported country and in a supported language. The country and language can be configured by setting the appropriate parameters in the GNConfig object provided when invoking a GNOperations method. These parameters are described below.

Parameter	Description	Default
content.country	Specifies the country of the delivered genre descriptors (using ISO county code)	null. Prior to version 2.5.8, it was USA.
content.lang	Specifies the language of the delivered genre descriptors (using ISO lang code)	""

See Localization and Internationalization for more information.

Accessing Genres

Genre data is delivered via a result-ready object in a GNDescriptor instance. Multiple genre levels can be delivered for a single album or track, so a collection of GNDescriptor objects is delivered.

When track-level genre descriptors are requested, Mobile Client returns them, if they are available. If track-level descriptors are not available, album-level descriptors are returned instead.

The delivered genre levels are stored in an array of GNDescriptor objects. The order of the GNDescriptor objects in the collection is representative of their level.

Genres are defined by a descriptor and an identifier. The descriptor is a word or phrase describing the genre. The descriptor string should only be used to display to the end user. Because descriptor names and user language settings are subject to change, the identifier is best suited for dealing with genre associations in a programmatic fashion.

The GNDescriptor object is fully described in the data dictionary, see GNDescriptor.

The following code snippet shows how genres can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        NSArray* trackMood = bestResponse.trackGenre;
        NSArray* albumMood = bestResponse.albumGenre;
// Display or store album and track genre as desired
    }
}
@ end
```

Mood

Mobile Client can return track level mood descriptors with a recognition or search result. Mood descriptors can be displayed to the end-user or can be used to categorize music in the user's collection for organization, navigation, or playlist generation.

Your application must be entitled to retrieve mood data.

A Mobile Client operation can be configured to include mood descriptors in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.mood	Set to true to enable retrieval of mood descriptors	false
content.mood.level	Sets the option for the mood descriptors returned; DEFAULT and EXTENDED	DEFAULT

Mood Levels

Gracenote has multiple levels of mood descriptors. Each level describes the related music with a different amount of detail and granularity.

Two options for mood are available, DEFAULT and EXTENDED. The DEFAULT option returns a single, default mood descriptor in the result. The EXTENDED option returns multiple levels of mood descriptors in the result.

The recommended option to use depends upon the application, its use cases, and the target audience. Some applications might want to use the DEFAULT option, as it is the simplest and contains the most commonly known moods. Other applications might want to provide a richer experience via the EXTENDED option. Applications might also use the the EXTENDED option to give their users a choice of which mood level is used.

Mood Localization

Gracenote provides a global solution and Mobile Client can be configured to deliver moods in a supported language. The language can be configured by setting the appropriate parameter in the GNConfig object provided when invoking a GNOperation. These parameters are described below.

Parameter	Description	Default
content.lang	Specifies the language of the delivered mood descriptors	""

See Localization and Internationalization for more information.

Accessing Moods

Mood data is delivered via a result-ready object in a GNDescriptor instance. Multiple mood levels may be delivered for a single track, so a collection of GNDescriptor objects is delivered.

The delivered mood levels are stored in an array of GNDescriptor objects. The order of the GNDescriptor objects in the collection is representative of their level.

Moods are defined by a descriptor and an identifier. The descriptor is a word or phrase describing the mood. The descriptor string should only be used to display to the end user. Because descriptor names and user language settings are subject to change, the identifier is best suited for dealing with mood associations in a programmatic fashion.

The GNDescriptor object is fully described in the data dictionary, see GNDescriptor.

The following code snippet shows how mood data can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
```

```
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        NSArray* trackMood = bestResponse.mood;
// Display or store track mood as desired
    }
}
@ end
```

Tempo

Mobile Client can return track-level tempo data with a recognition or search result. Tempo data can be displayed to the end-user or can be used to categorize music in the user's collection for organization, navigation, or playlist generation.

Your application must be entitled to retrieve tempo data.

A Mobile Client operation can be configured to include tempo data in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.tempo	Set to true to enable retrieval of tempo descriptors	false
content.tempo.level	Sets the option for the tempo descriptors returned; DEFAULT and EXTENDED	DEFAULT

Tempo Levels

Gracenote has multiple levels of tempo descriptors. Each level describes the related music with a different amount of detail and granularity.

Two options for tempo are available: DEFAULT and EXTENDED. The DEFAULT option returns a single, default tempo descriptor in the result. The EXTENDED option returns multiple levels of tempo descriptors in the result.

The recommended option to use depends upon the application, its use cases, and the target audience. Some applications might want to use the DEFAULT option, and other applications might want to provide a richer experience via the EXTENDED option. Applications might also use the EXTENDED option to give their users a choice of which tempo level is used.

Tempo Localization

Gracenote provides a global solution and Mobile Client can be configured to deliver tempo in a supported language. The language can be configured by setting the appropriate parameter in the GNConfig object provided when invoking a GNOperation. These parameters are described below.

Parameter	Description	Default
content.lang	Specifies the language of the delivered tempo descriptors	""

See Localization and Internationalization for more information.

Accessing Tempo Data

Tempo data is delivered via a result-ready object in a GNDescriptor instance. Multiple tempo levels may be delivered for a single album or track, so a collection of GNDescriptor objects is delivered.

The delivered tempo levels are stored in an array of GNDescriptor objects. The order of the GNDescriptor objects in the collection is representative of their level.

Tempos are defined by a descriptor and an identifier. The descriptor is a word or phrase describing the tempo. The descriptor string should only be used to display to the end user. Because descriptor names and user language settings are subject to change, the identifier is best suited for dealing with tempo associations in a programmatic fashion.

The GNDescriptor object is fully described in the data dictionary, see GNDescriptor.

The following code snippet shows how tempo data can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        NSArray* trackTempo = bestResponse.tempo;
// Display or store track tempo as desired
    }
}
@ end
```

Origin, Era, and Artist Type

Mobile Client can return origin, era, and artist type data with a recognition or search result. Origin data gives the geographic location most strongly associated with the artist. Era data gives the time period most strongly associated with the artist. Artist type data gives the gender and composition (solo, duo, group) of the artist. Origin, era, and artist type data can be displayed to the end-user or can be used to categorize music in the user's collection for organization, navigation, or playlist generation.

A Mobile Client operation can be configured to include origin, era, and artist type data in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.origin	Set to true to enable retrieval of origin descriptors	false
content.era	Set to true to enable retrieval of era descriptors	false
content.artistType	Set to true to enable retrieval of artist type descriptors	false

Origin, era, and artist type data is delivered as a bundle. If the value of any of the parameters is set to TRUE, all three types of descriptors are delivered. To turn off delivery of the descriptors, all three parameters must be set to FALSE.

Origin, Era, and Artist Type Levels

Gracenote has multiple levels of origin, era, and artist type descriptors. Each level describes the related music with a different amount of detail and granularity. The following GNConfig parameters set the option for the origin, era, and artist type descriptors returned:

content.origin.level	Sets the option for the origin descriptors returned; DEFAULT and EXTENDED	DEFAULT
content.era.level	Sets the option for the era descriptors returned; DEFAULT and EXTENDED	DEFAULT
content.artistType.level	Sets the option for the artist type descriptors returned; DEFAULT and EXTENDED	DEFAULT

Two options for origin, era, and artist type descriptors are available: DEFAULT and EXTENDED. The DEFAULT option returns a single, default descriptor in the result. The EXTENDED option returns multiple levels of descriptors in the result.

The recommended option to use depends upon the application, its use cases, and the target audience. Some applications might want to use the DEFAULT option, and other applications might want to provide a richer experience via the EXTENDED option. Applications might also use the EXTENDED option to give their users a choice of which level is used.

Origin, Era, and Artist Type Localization

Gracenote provides a global solution and Mobile Client can be configured to deliver origin, era, and artist type in a supported language. The language can be configured by setting the appropriate parameter in the GNConfig object provided when invoking a GNOperation. The parameter is described below.

Parameter	Description	Default
content.lang	Specifies the language of the delivered tempo descriptors	""

See Localization and Internationalization for more information.

Accessing Origin, Era, and Artist Type Data

Origin, era, and artist type data is delivered via a result-ready object in a GNDescriptor instance. Multiple levels may be delivered, so a collection of GNDescriptor objects is delivered.

The delivered levels are stored in an array of GNDescriptor objects. The order of the GNDescriptor objects in the collection is representative of their level.

Origin, era, and artist type are defined by a descriptor and an identifier. The descriptor is a word or phrase describing the origin, era, or artist type. The descriptor string should only be used to display to the end user. Because descriptor names and user language settings are subject to change, the identifier is best suited for dealing with origin, era, and artist type associations in a programmatic fashion.

The GNDescriptor object is fully described in the data dictionary, see GNDescriptor.

The following code snippet shows how origin, era, and artist type data can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        NSArray* originDescriptorsArray = bestResponse.origin;
        NSArray* eraDescriptorsArray = bestResponse.era;
        NSArray* artistTypeDescriptorsArray = bestResponse.artistType;
// Display or store origin, era, and artist type as desired
    }
}
@ end
```

Retrieving Cover Art

Mobile Client can return album Cover Art with a recognition or search result. Cover Art can be used effectively to enrich the user experience.

Note that only the cover art URL is returned with the result. To get the actual image data, your app needs to invoke the GNCoverArt.data accessor method, which should be done in a background thread. See the Sample App for an example of getting cover art data in a background thread.

Your application must be entitled to retrieve Cover Art.

A Mobile Client operation can be configured to include Cover Art in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default

content.coverArt	Set to true to enable retrieval of cover art.	false (as of version 2.5.8)
content.coverArt.sizePreference	Comma-separated list of cover art size preferences. Also specifies size preference for artist images.	"SMALL,MEDIUM, THUMBNAIL, LARGE,XLARGE"

The Cover Art size preference takes a comma-separated list of the preferred Cover Art sizes, in order of preference. Mobile Client will return the first Cover Art image that matches a size in the preference list. If a size is not in the preference list, it will not be returned.

If an invalid size preference is included in the list, Mobile Client ignores the list and uses the default list instead.

For more information on using images returned by Mobile Client see Image Resizing Best Practice.

You can improve retrieval performance for some operations by retrieving cover art in the background. For more information, see Improving Retrieval Performance by Using Enriched Content URLs.

Genre Cover Art

In cases where Gracenote does not have cover art for a particular album, genre-themed artwork is instead returned in a response. This option can be disabled on a per-query basis by setting the appropriate GNConfig parameter as shown below.

Parameter	Description	Default
content.coverArt.genreCoverArt	Set to true to receive genre cover art when album cover art is not available	true

To receive genre art, both content.coverArt.genreCoverArt and content.coverArt must be set to true. Your application also must be entitled for cover art.

The size of the Genre Cover Art returned is governed by the Cover Art size preference GNConfig parameter described above.

For more information on using images returned by Mobile Client see Image Resizing Best Practice.

Accessing Cover Art

Cover Art is delivered via a result-ready object as a GNCoverArt object. Cover Art is provided as a raw data in NSdata datatype that can be easily used to draw an image.

The GNCoverArt object is fully described in the data dictionary, see GNCoverArt.

The following code snippet shows how tempo data can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        GNCoverArt* coverArt = bestResponse.coverArt;
// Display or store cover art as desired
    }
}
@ end
```

Artist Images

Mobile Client can return an Artist Image with a recognition or search result. Artist Images can be used effectively to enrich the user experience.

Your application must be entitled to retrieve Artist Images.

A Mobile Client operation can be configured to include an Artist Image in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.contributor.images	Set to true to enable retrieval of artist images	false
content.coverArt.sizePreference	Comma-separated list of artist image size preferences. Also specifies size preference for cover art images.	"SMALL,MEDIUM, THUMBNAIL, LARGE,XLARGE"

The Artist Image size preference takes a comma-separated list of the preferred Artist Image sizes, in order of preference. Mobile Client returns the first Artist Image that matches a size in the preference list. If a size is not in the preference list, it will not be returned.

For more information on using images returned by Mobile Client see Image Resizing Best Practice.

You can improve retrieval performance for some operations by retrieving artist images in the background. For more information, see Improving Retrieval Performance by Using Enriched Content URLs.

Accessing Artist Images

Artist Images are delivered via a result-ready object as a NSData. The raw data can be easily used to draw an image.

The following code snippet shows how artist images can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        GNImage artistImage = bestResponse.contributorImage;
        NSData* artistImageData = artistImage.data;
// Display or store artist image as desired
    }
}
@ end
```

Artist Biographies

Mobile Client can return an Artist Biography with a recognition or search result. Artist Biographies can be used effectively to enrich the user experience.

Your application must be entitled to retrieve Artist Biographies.

A Mobile Client operation can be configured to include an Artist Biography in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.contributor.biography	Set to true to enable retrieval of artist biographies	false

You can improve retrieval performance for some operations by retrieving artist biographies in the background. For more information, see Improving Retrieval Performance by Using Enriched Content URLs.

Accessing Artist Biographies

Artist Biographies are delivered via a result-ready object as a collection of String objects, where each String represents an individual paragraph of the biography.

The following code snippet shows how an artist biography can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        NSArray* artistBiography = bestResponse.artistBiography;
```

```
// Display or store artist biography as desired
    }
}
@ end
```

Album Reviews

Mobile Client can return an Album Review with a recognition or search result. Album Reviews can be used effectively to enrich the user experience.

Your application must be entitled to retrieve Album Reviews.

A Mobile Client operation can be configured to include Album Reviews in its response, via the GNConfig object provided when the operation is invoked. These parameters are described below.

Parameter	Description	Default
content.review	Set to true to enable retrieval of album reviews	false

You can improve retrieval performance for some operations by retrieving album reviews in the background. For more information, see Improving Retrieval Performance by Using Enriched Content URLs.

Accessing Album Reviews

Album Reviews are delivered via a result-ready object as a collection of String objects, where each String represents an individual paragraph of the review.

The following code snippet shows how an album review can be retrieved from a result-ready object.

```
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        NSArray* albumReview     = bestResponse.albumReview;
// Display or store album review as desired
    }
}
@ end
```

Retrieval Methods

Related content can be retrieved from the 2.5.9.8 returned by an audio recognition operation or via a Gracenote unique album or track identifier.

Retrieving Related Content for a Single Best Match

A single best match result returned by an audio recognition operation can contain related content. The following methods can be used to retreive the respective content:

- GNSearchResponse:coverArt
- GNSearchResponse:contributorImage
- GNSearchResponse:getArtistBiography
- GNSearchResponse:getAlbumReview

The GNConfig object provided to the recognition operation must first be configured to retrieve related content using the configuration parameters shown above. The following code example illustrates how to configure the configuration object, initiate a recognition event, and retrieve the related content from the returned result:

```
// Set configuration properties
[config setProperty: @"content.coverArt" value: @"1"];
[config setProperty: @"content.contributor.images" value: @"1"];
[config setProperty: @"content.contributor.biography" value: @"1"];
[config setProperty: @"content.review" value: @"1"];
// Define result-ready object to extract related content from recognition
result
@ interface ApplicationSearchResultReady : NSObject <GNSearchResultReady>
{
}
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestResponse = [result bestResponse];
    if (bestResponse != nil)
    {
// Extract related content from response
        GNCoverArt* coverArt     = bestResponse.coverArt;
        NSData*  artistImage     = bestResponse.contributorImage;
        NSArray* artistBiography = bestResponse.artistBiography;
        NSArray* albumReview     = bestResponse.albumReview;
// Display related content as desired
    }
}
@ end
// Use configuration object when invoking a recognition operation
- (void) recognizeFromMic: (GNConfig*) config
{
```

```
// Create result-ready object to receive retrieval result and extract
related content
    ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
// Invoke recognition operation
    [GNOperations recognizeMIDStreamFromMic: searchResultReady config: con-
fig];
}
```

For subsequent operations after an initial identification, retrieving related content from a Gracenote album or track identifier is more efficient than repeating the full recognition process. To minimize response time, extract and store the Gracenote Identifier after the initial recognition operation and use it for subsequent retrieval operations, as shown in the next section.

Retrieving Related Content by Gracenote Identifier

Gracenote unique album and track identifiers can be obtained from any Gracenote product, enabling Mobile Client to be easily integrated into a larger music identification system. The following code example shows how to obtain a Gracenote Identifier for a specific album and use it to retrieve related content; the same technique can be used to retrieve related content for a track identifier instead of an album identifier, by using the methods GNSearchResponse:fetchRelatedContent:forTrackId and GNOperations:fetchByTrackId:config:trackId: instead of GNSearchResponse:fetchRelatedContent:forAlbumId and GNOperations:fetchByAlbumId:config:albumId:

```
// Result-ready object to extract related content from recognition result
by album identifier:
// Provide implementation for GNResultReady method
@ implementation ApplicationGetGnIdResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestresponse = [result bestResponse];
    if (bestresponse != nil)
    {
// Extract album identifier and store for later use
        gnAlbumID = bestResponse.albumId;
// Retrieve related content using album identifier
        [self fetchRelatedContent: self.config forTrackId: bes-
tresponse.trackId];
    }
}
@ end
// Initial recognition request; result will contain a Gracenote album iden-
tifier
// Create result-ready object to receive search result and extract and
store track identifier and related content
ApplicationGetGnIdResultReady* getGnIdResultReady = [Appli-
cationGetGnIdResultReady alloc];
```

```
// Invoke recognition operation
[GNOperations recognizeMIDStreamFromMic: getGnIdResultReady config: con-
fig];
// Later retrieval request using previously stored album identifier
- (void) fetchRelatedContent: config forAlbumId: albumId
{
// Set configuration properties
    [config setProperty: @"content.contributor.images" value: @"1"];
    [config setProperty: @"content.contributor.biography" value: @"1"];
    [config setProperty: @"content.review" value: @"1"];
// Create result-ready object to receive retrieval result
    ApplicationSearchResultReady* searchResultReady = [Appli-
cationSearchResultReady alloc];
    [GNOperations fetchByAlbumId: searchResultReady config: config trackId:
albumId];
}
// Define result-ready object to extract related content from recognition
result:
// Provide implementation for GNResultReady method
@ implementation ApplicationSearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
// Get best response from search result
    GNSearchResponse* bestresponse = [result bestResponse];
// Extract related content from response
    if (bestresponse != nil)
    {
        NSArray* albumReview     = bestResponse.albumReview;
        NSArray* artistBiography = bestResponse.artistBiography;
        GNImage*  artistImage     = bestResponse.contributorImage;
    }
}
```

During the retrieval process, Mobile Client sends status updates to notify the application as each of the stages shown in the figure is reached:

Status flow in related content retrieval by Gracenote Identifier

Track and Album Identifiers can be retrieved from any of Gracenote's products, enabling Mobile Client to be easily integrated into a larger music identification system.

Retrieving Link Data

When an application is appropriately entitled, Mobile Client returns Link identifiers for all responses provided in a result-ready Object instance. The identifiers may be sourced from a third party (such as Amazon.com), or they may be specific to your organization. You can use such Link data, for example, to

redirect the user to a site (such as Amazon) where they can purchase the matched track, or as a key into your own data catalog. These examples demonstrate coding this functionality:

```
// Result-ready object implements GNFingerprintResultReady protocol
@ interface SearchResultReady : NSObject <GNFingerprintResultReady>{
}
@ end
// Provide implementation for GNResultReady method
@ implementation SearchResultReady
- (void) GNResultReady: (GNSearchResult*) result
{
    GNSearchResponse* bestresponse = [result bestResponse];
    if (response != nil)
    {
        NSArray linkItems = [response albumLinkData];
        if (linkItems != nil)
        {
            NSEnumeration* e = [linkItems objectEnumerator];
            id object;
            while (object = [e nextObject])
            {
  // Use link data as desired
            }
        }
    }
}
@ end
```

You can tune audio recognition events to prefer results containing Link identifiers from a specific source, by setting the content.link.preferredSource property of the GNConfig object used when invoking the audio recognition method, as shown here:

```
// Set the preferred source property to prefer results that contain Link
IDs
// from a specific source
GNConfig* config = [GNConfig init: @"12345678-ABCDEFGHIJKLM-
NOPQRSTUVWXYZ012345"];
[config setProperty: @"content.link.preferredSource" value: @"Amazon"];
[GNOperations recognizeMIDStreamFromMic: searchResultReadyObject config:
config];
```

Gracenote Web Services returns the best available match containing a Link identifier from that source, if available. Note, however, that the single best match returned is not necessarily guaranteed to contain a Link identifier from the preferred source, if none of the available matches contains one from that source.

While commerce identifiers can be requested, AlbumID does not support the preference of a specific identifier over the actual Album a song came from. These preferred results can instead be obtained by using GNOperations.recognizeMIDFileFromFile or GNOperations.recognizeMIDFileFromPcm. See MusicID-File for more information.

Status Change Updates

When performing an operation, Mobile Client sends status updates to the application via the GNOperationStatusChanged protocol.

To receive status changed updates, the application must create a class that implements a result-ready protocol and the GNOperationStatusChanged. An instance of the object must be provided when the operation is invoked.

When an application receives status changed updates, the user can be notified on the progress of the operation.

For deterministic functions within an operation, the percent completed is also provided. This is currently only provided for recording audio from the microphone. These examples demonstrate coding this functionality:

```
// Result-ready object implements GNSearchResultReady and
// GNOperationStatusChanged protocols
@ interface SearchResultsStatusReady : NSObject <GNSearchResultReady, GNOp-
erationStatusChanged>
{
}
@ end
// Provide implementation for GNOperationStatusChanged method
@ implementation SearchResultsStatusReady
// Method to handle the status changed updates from the operation
- (void) GNStatusChanged: (GNStatus*) status
{
    NSString* msg;
    if (status.result == RECORDING)
    {
        msg = [NSString stringWithFormat: @"%@ %d@", status.message, stat-
us.percentDone, @"%"];
    }
    else
    {
        msg = status.message;
    }
// Display status message to user
}
// Method to handle result returned from operation
- (void) GNResultReady: (GNSearchResult*) result
{
}
@ end
// Create the result-ready object to receive the recognition results then
invoke
// the operation
SearchResultsStatusReady* resultready = [SearchResultsStatusReady alloc];
[GNOperations recognizeMIDStreamFromMic: resultready config: config];
```

Canceling an Operation

Mobile Client supports canceling a running operation. To cancel an operation, the application must call the GNOperations:cancel method with the same result-ready object instance that was provided to the operation when it was invoked. Mobile Client uses the result-ready object instance to identify which operation to cancel. These examples demonstrate coding this functionality:

```
// Create the result-ready object and invoke the operation
SearchResultsStatusReady* resultready = [SearchResultsStatusReady alloc];
[GNOperations recognizeMIDStreamFromMic: resultready config: config];
// Use the result-ready object provided when invoking the operation to can-
cel
// the operation
[GNOperations cancel: resultReady];
```

When GNOperations:cancel is called, the associated operation is stopped. It is not always possible to cancel an operation immediately; the operation may need to complete blocking functions before it can completely cancel. Blocking functions are functions that cannot be interrupted. Some of these functions can also block other operations from continuing.

Important: An operation may not be instantaneously canceled when GNOperations:cancel is called.

Localization and Internationalization

Gracenote provides a global solution and Mobile Client can be configured to deliver metadata that is specific to a supported region or a supported language. A country or language can be configured by setting the appropriate GNConfig parameters. The parameters are described below.

Parameter	Description
content.country	Specifies the country of the delivered metadata. Default for this is null (as of version 2.5.8). Prior to this, it was USA
content.lang	Specifies the language of the delivered metadata

The specified country and language can affect the delivered metadata in various ways:

1. When a supported country is specified, the genre descriptors for that country are delivered
2. When a supported language is specified, album and track metadata are delivered in that language, if available
3. When a supported language is specified, genre, mood, tempo, origin, era, and artist type descriptors are delivered in that language, if available

For a complete list of supported languages see the Mobile Client Reference Guide.

All country codes specified by ISO 3166-1 alpha-3 are supported.

The iOS Developers Guide also provides information on supporting localization and internationalization.

Using Mobile Client Debug Logging

Mobile Client can be configured to output useful debugging information to the console. To enable debugging, set the debugEnabled property to "1", "true", or "True" in a GNConfig object instance, and provide that instance when invoking an operation, as shown in the example. Note that only operations that are passed a GNConfig object instance with debugging enabled will generate logging output.

```
[config setProperty: @"debugEnabled" value: @"1"];
```

The debug log feature is provided for development only. Production applications cannot have debug log generation enabled or provide an option to enable it.

During validation, Gracenote ensures debug log generation is not and cannot be enabled by the application.

Data Dictionary

This section describes the different data fields returned from Mobile Client. Two different types of results can be generated by Mobile Client: a music search result and a fingerprint creation result. Both result types are described in the sections below.

Search Result

A Mobile Client search result returns a variety of metadata fields that can be used to enrich the user experience. The following sections contain a description of each field and the hierarchy in which the fields are delivered to your application.

GNSearchResult

Contains the result(s) of a search operation.

Field	Description	Type	Accessor	Comments

Responses	Collection of albums that contain tracks that were matched by the search operation	NSArray	responses	
Best Response	Album that contains the track that is the best match of the search operation	GNSearchResponse	bestResponse	

GNSearchResponse

Describes the metadata fields for an album and the track on that album that was matched by the search operation.

Field	Description	Type	Accessor	Comments
Album Artist Name	Name of the album level artist	NSString	albumArtist	
Album Artist Name Betsumei	Japanese alternate names and pronunciations for album level artist name	NSString	albumArtistBetsumei	
Album Artist Name Yomi	Japanese phonetic representation of album level artist name	NSString	albumArtistYomi	
Album Genre	Album level genre descriptors	NSArray	albumGenre	
Album ID	Gracenote unique identifier for the album	NSString	albumId	
Album Link Data	Link identifiers related to the album	NSArray	albumLinkData	albumLinkData can be nil

Album Release Year	Year the album was released in	NSString	albumReleaseYear	
Album Review	Album review	NSArray	albumReview	albumReview may or may not be nil
Album Title	Title of the album	NSString	albumTitle	
Album Title Yomi	Japanese phonetic representation of Album Title	NSString	albumTitleYomi	
Album Track Count	Number of tracks on the album	NSInteger	albumTrackCount	-1 if no track
Artist Biography	Artist's biography	NSArray	artistBiography	artistBiography may or may not be nil
Artist Era	Era descriptors	NSArray	era	
Artist Image	Image of artist who contributed in the album	GNImage	contributorImage	contributorImage may or may not be nil
Artist Origin	Origin descriptors	NSArray	origin	
Artist Type	Artist type descriptors	NSArray	artistType	
Cover Art	Album cover art	GNCoverArt	coverArt	When cover art is not available, genre art may be returned (based on availability)
Track Artist Name	Name of the album artist	NSString	artist	
Track Artist Name Betsumei	Japanese alternate names and pronunciations for Artist name	NSString	artistBetsumei	

Track Artist Name Yomi	Japanese phonetic representation of Artist name	NSString	artistYomi	
Track Genre	Track genre	NSarray	trackGenre	genre can be nil
Track ID	Gracenote unique identifer for the track	NSString	trackId	trackId can be nil in the case of Lyric_search
Track Link Data	Link identifiers related to the track	NSArray	trackLinkData	trackLinkData can be nil
Track Match Position	Indicates at what timeslice within the song the audio sample was matched	NSString	songPosition	Unit is in milliseconds; only applies to MusicID-Stream
Track Mood	Track level mood descriptors	NSArray	mood	
Track Number	One-based index of the track on the album	NSInteger	trackNumber	-1 if no trackNumber
Track Tempo	Track level tempo descriptors	NSArray	tempo	
Track Title	Title of the track	NSString	trackTitle	
Track Title Yomi	Japanese phonetic representation of Track Title	NSString	trackTitleYomi	

GNAlbumIdSearchResult

Contains the result(s) of an albumId search operation.

Field	Description	Type	Accessor	Comments
No Match Responses	AlbumId results with No_Match	NSMutableArray	noMatchResponses	

Error Responses	AlbumId results with GNAlbumIDFileError	NSMutableArray	errorResponses	

GNAlbumIdSearchResponse

Describes the metadata fields for an album and the track on that album that was matched by the albumId search operation. Inherited from GNSearchResponse. Contains all GNSearchResponse objects and fileIndent used in albumId operation per file.

Field	Description	Type	Accessor	Comments
GNSearchResponse	All GNSearchResponse metadata			Inherited from GNSearchResponse
File Ident	File Identifier	NSString	fileIdent	

GNDescriptor

Defines a Gracenote descriptor. The genre, mood, tempo, origin, era, and artist type data delivered by Mobile Client is referred to as descriptors. This object describes a descriptor by providing its name and identifier.

Field	Description	Type	Accessor	Comments
Descriptor	Descriptor name	NSString	itemData	
ID	Descriptor identifier	NSString	itemId	

GNAlbumIdFileError

Container class for the errors that can occur in AlbumId operations.

Field	Description	Type	Accessor	Comments
Error Message	Indicates results with error	NSString	errMessage	
Error Code	Error code for file	NSString	errCode	
File Indent	File identifier	NSString	fileIdent	

GNLinkData

Holds a collection of Link identifiers and their providers.

Field	Description	Type	Accessor	Comments

Id	Actual value from the provider	NSString	uid	
Source	The provider of the Link identifier, such as "Amazon"	NSString	source	

GNCoverArt

Contains the cover art image for an album.

Field	Description	Type	Accessor	Comments
Data	Cover art image as a data stream	NSData	data	
MIME Type	Cover art data type. This can be used to appropriately read and render the cover art image.	NSString	mimeType	
Size	The size of the cover art returned as THUMB, SMALL, MEDIUM, LARGE or XLARGE. For the size in pixels see Image Resizing Best Practice.	NSString	size	

GNImage

Contains the contributor image for an album.

Field	Description	Type	Accessor	Comments
Data	Contributor image as a data stream.	byte[]	getData	
MIME Type	Contributor data type. This is used to read and render the contributor image appropriately.	String	getMimeType	
Size	The size of the contributor image returned as THUMB, SMALL, MEDIUM, LARGE or XLARGE. For the size in pixels see Image Resizing Best Practice.	String	getSize	

Fingerprint Creation Result

A Mobile Client fingerprint creation result delivers the created fingerprint to your application in a simple hierarchy described below.

GNFingerprintResult

Field	Description	Type	Accessor	Comments
Fingerprint Data	Fingerprint generated by operation. The fingerprint can be used to recognize the audio.	NSString	fingeprintData	The fingerprint data can be sent to other Gracenote products for other uses.

Considerations

Recognizing Audio from the Microphone

The following practices are recommended when recognizing audio from the microphone:

- Provide clear and concise onscreen instructions on how to record the audio:
    - Most failed recognitions are due to incorrect operation by the user.
    - Clear and concise instructions help the user correctly operate the application, resulting in a higher match rate and a better user experience.
- While recording audio from the microphone display a progress indicator:
    - When Mobile Client is recording from the microphone, the application can receive status updates. The status updates indicate what percentage of the recording is completed.
    - The status updates are issued at every tenth percentile of completion, meaning 10%, 20%, 30%, and so on.
    - Use this information to display a progress bar (indicator) to notify the user.
    - When recording has finished use vibration or a tone (or both) to notify the user.
- Visual only notifications can hamper the user experience because:
    - The user may not see the notification if they are holding the handset up to an audio source.
    - The user may pull the device away from an audio source to check if recording has completed. This may result in a poor quality recording.
    - While Web Services is being contacted, display an animation that indicates the application is performing a function. If the application appears to halt the user

may believe the application has crashed.

- If no match is found, reiterate the usage instructions and ask the user to try again.

Recognizing Audio from a File using MusicID-File

The following practices are recommended when recognizing audio from a file when using MusicID-File:

- Allow the user to select multiple files for recognition. Submit them concurrently for fastest results.
- Limit the number of stored results and pending recognitions:
    - File recognition operations can run concurrently, which allows many requests to be issued at once.
    - Each recognition operation and each returned result requires memory.
    - Be careful to limit the number of results stored and pending recognition operations, to avoid exhausting the device's RAM.
- While Web Services is being contacted, display an animation that indicates the application is performing a function. If the application appears to halt, the user may believe the application has crashed.

Image Resizing Best Practice

Summary

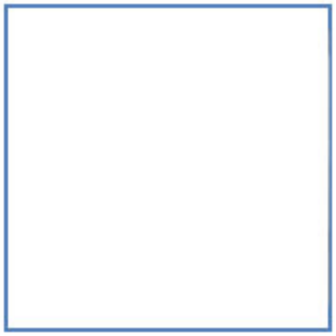
This topic provides guidance on image implementation by discussing Gracenote's use of predefined square dimensions to accommodate variances among image orientations.

Description

Gracenote provides a variety of images as part of our enhanced content offerings. For Mobile Client, these include Cover Art, Artist, and Genre images.

To accommodate variation in the images' dimensions, we resize the images to fit within standardized image dimensions a predefined-square so that the longest dimension equals the dimensions of one side of the square, while the other dimension is somewhat short of the full square. The following images show how an image is resized within the predefined square. (Note that the outline for the square is used here only to demonstrate layout. We do not recommend displaying this outline on your application's user interface.)

Predefined Square



Horizontally-oriented Image Examples

Vertically-oriented Image Examples

The following sections discuss the differences between music images and give the standard image dimensions for each image type.

Music Cover Art

While CD cover art is often represented by a square, it is more accurately a bit wider than it is tall. The dimensions of these cover images vary from album to album. Some CD packages, such as a box set, might even be a radically different shape.

Gracenote Standard Image Dimensions: Music Cover Art

Size	W x H (Pixels)
Thumbnail	75 x 75
Small	170 x 170
Medium	450 x 450
Large	720 x 720
Extra Large	1080 x 1080

Artist Images

Like music Cover Art, Artist images are seldom square, and are generally more obviously rectangular than music cover art. Because music artists range from solo performers to bands with many members, the images may either be wide or tall.

Gracenote Standard Image Dimensions: Artist Images

Size	W x H (Pixels)
Thumbnail	75 x 75
Small	170 x 170
Medium	450 x 450
Large	720 x 720
Extra Large	1080 x 1080

Image Implementation Recommendation

For all images: We recommend that the image be centered horizontally and vertically within the predefined square dimensions, and that the square be transparent so that the background shows through. This results in a consistent presentation despite variation in the image dimensions.

Searching by Text, Lyric Fragment, and Fingerprint

The following practices are recommended when searching by text, lyric fragment, and fingerprint.

- Allow user to input as much information as possible:
    - Text search allows artist name, album title and track title to be entered. Allow the user to input all of these fields and pass them to Gracenote for the search
    - Lyric Fragment search can be augmented with the artist name. Allow the user to provide the artist name and ensure it is passed to Gracenote for the search.
- While Web Services is being contacted display an animation that indicates the application is performing a function. If the application appears to halt the user may believe the application has crashed.

Canceling

The following practices are recommended when canceling an operation.

- After calling cancel on an operation ignore all status updates received for that operation.
    - The operation may run and, possibly, issue status updates until the blocking functions are completed. In this situation, the status updates should be ignored.
- Display the word Canceling to the user while waiting for a canceled operation to complete blocking functions.
- Consider canceling while recording from the microphone:
    - A microphone recognition operation is invoked by the user.
    - The application starts recording audio and the user invokes cancel.
    - Recording may take a one or two seconds to stop.
    - Display Canceling until the canceled operation has completed the recording function
    - While canceling, the user should be prevented from invoking a new microphone recognition operation

Retry Facility

Occasionally Mobile Phone Handsets are unable to successfully create a network connection. This could be because the Mobile Phone Handset is not within range of a suitable network or the network is temporarily unavailable due to device or network limitations.

It is recommended that you implement a retry facility that retries the user request several times before reporting to the user that the request could not be fulfilled. A retry facility allows temporary network unavailability to go unnoticed by the user.

Resource Management

Mobile Client is a compact library that provides easy access to rich content that can enhance the user's experience. Its compact size and simplicity are not indicative of the vast amount of content it can quickly deliver.

It is important that your application is prepared to correctly manage requests for content and is able to handle the content when it is delivered.

Storing Content in RAM

The richness of the content delivered by Mobile Client and the speed of delivery can easily result in megabytes of data being transferred to the device. This can quickly consume the RAM resources available to applications, especially those operating in a resource-limited environment.

Applications must consider carefully how data is stored and how long it is stored.

If your application performs a single track recognition and then displays the result, the RAM resources are unlikely to be exhausted. But if the application performs thousands of track recognitions and intends to display all the returned data, then RAM usage will become an issue.

Applications should limit the amount of data they attempt to store in RAM. For example, an application that recognizes a user's entire song collection could limit the number of recognition results using paging. The application can recognize a small number of songs, display the results for the user to peruse, and then when prompted by the user, recognize the next set of songs. For more information, see Result Paging.

Applications should also consider what needs to be stored in RAM. It might be acceptable for your application to store data returned from Mobile Client directly into persistent memory, only using RAM to display small subsets of the data to the user.

The schemes used to manage RAM resources will vary greatly with use case and implementation. It is important that dealing correctly with data delivered from Mobile Client be a major consideration for the development team early in the design phase.

Request Flow Control

Mobile Client is capable of processing multiple requests concurrently. This facility allows an application to obtain results faster than issuing one query at a time.

There is no limit to the number of requests that can be issued; Mobile Client queues the requests until they can be processed. Each request, whether it is queued or being processed, consumes RAM. If requests are issued faster than they can be processed, the application may run out of memory.

To avoid this situation an application should implement flow control for issuing requests. The number of requests pending (queued or being processed) should be capped at a maximum. It is recommended that you limit the number of pending requests to five.

Applications should issue requests until the number of pending requests reaches the predefined maximum. No further requests should be issued until the number of pending requests drops below the predefined maximum.

In addition, all layers in the application stack should implement request flow control, or be capable of rejecting requests from upper layers.

For example, if you are developing a middleware layer that interfaces upper layer applications with Mobile Client, your middleware layer should implement flow control. If the number of pending requests rises to the predefined maximum, the layer should reject any new requests until the number of pending requests drops below the predefined maximum. Sophisticated applications will also provide notifications to upper layers when requests can and can't be accepted (similar to a modem's XON and XOFF commands).

Result Paging

In addition to request flow control, an application with user navigable results should optimally use results paging to limit the number of requests issued and results stored. Paging can also be used to ensure the application is responsive.

Results paging can be effective in applications that can display a large number of results in sections, such as a scrolling display, or paged results where Next and Previous controls are used for result navigation.

It is recommended that such applications store the result for three (3) pages: the current page, the previous page and the next page. As the user navigates through the pages, the results in the current, previous, and next pages change. As they change the application can delete stored results and retrieve new results as they are needed to fill the current, previous, and next pages.

Using this scheme the previous and next pages are pre-retrieved, making the application responsive as the user navigates.

Limiting the total number of results pre-retrieved also limits the amount of content that is stored in RAM. Gracenote recommends using five results per page. If the user is limited to moving one results page at a time, only one page of results needs to be retrieved with each move. This limits the number of concurrent queries to five, as recommended in Request Flow Control.

Error Handling

When an operation cannot be completed, Mobile Client generates the appropriate error information and returns it via a result-ready Object. Your application should handle the error conditions and

proceed accordingly.

In the event of an error, Mobile Client returns an error code and an error message. The error code can be used by your application to react to specific error conditions. The error message contains error condition information, which provides additional clues to the error's cause. Error messages are not suitable for display to the end-user, as they contain technical information and are English-language specific; they are also subject to change without warning. The error message displayed to the user should be derived from the error code.

Sophisticated implementations may provide an error notification mechanism that allows errors to be investigated in deployed applications. For example, your application can send error information to a support server or prompt the user to send error information to an email address. Once notification of an error has been received, technical personnel can investigate the error accordingly.

GNResult.FPXFailure and GNResult.FPXFingerprintingFailure

These errors are generated if Mobile Client cannot generate a fingerprint from an audio sample. Your application should indicate that the audio cannot be recognized due to an invalid or unsupported format.

GNResult.FPXRecordingFailure

This error is generated when Mobile Client cannot obtain the microphone for recording an audio sample. Your application should indicate to the user that the microphone is required to record music for identification and to try again after closing any applications that may be using the microphone.

GNResult.WSRegistrationInvalidClientIdFailure

This error is generated when the Client ID provided to Mobile Client is not successfully authenticated by Web Services. Your application should indicate to the user that the music identification service is unavailable.

GNResult.WSNetworkFailure

This error is generated when the device does not have a valid network connection; for example, a Mobile Phone Handset may not be within range of a suitable network. Your application should indicate that a working Internet connection is required and to try again when the connection is restored.

GNResult.WSFailure

This error is generated when Gracenote Web Services cannot fulfill a request from the Mobile Client. The accompanying error message is a description of the error condition as provided by Gracenote Web Services.

Your application should indicate that the user's request could not be fulfilled at the present time. Do not display the error message to the user, as it contains technical information that is generally not com-

prehensible to a user. Also, do not attempt to parse the message, as messages returned from Web Services change regularly without warning.

If your application provides an error notification mechanism, forward the error message to your technical personnel.

GNResult.WSInvalidDataFormatError

This error is generated when Mobile Client cannot parse the response received from Gracenote Web Services. This canoccur due to corruption of the results data. Your application should indicate to the user that their request failed and offer to retry the query.

GNResult.UnhandledError

This error code is reported when an unexpected error occurs that is outside the normal operation of Mobile Client and outside the scope of Mobile Client's error capture capabilities. The possible causes of such an error are very broad and include (but are not limited to) platform errors, third-party library errors, hardware errors, and other similar types of errors.

Your application should indicate that an error occurred and that the current request could not be completed.

Limiting Query Time

Mobile Client imposes time limits (timeouts) for the amount of time a device takes to create an Internet connection or fulfill an Internet request. These times can vary depending on:

- The device internal resources (memory, MIPS, and so on.)
- The capacity of the Internet connection (the amount of data that can be transported by the air interface)
- The availability of the Internet connection (devices may move in and out of signal range or the network may become congested)

The timeouts chosen by Mobile Client are based on the most limited device with the most limited Internet connection capacity and availability. Consequently, for high end devices, these timeouts may be considered too long.

It is recommended that devices impose their own query timeout mechanism that cancels a query when it exceeds an application specified time limit. This allows the application the flexibility to limit query lengths in accordance with the capabilities of the host devices.

When initiating a query to the Gracenote database via Mobile Client, start an application timer with an application specified timeout. If the timer expires before the query completes, cancel the query using the GNOperations.cancel() method.

When calling AlbumID operations, do not set a timer. AlbumID directory operations can take a long time, depending on many factors, such as the number of tracks in the directory, processor speed, network speed, type, and other factors.

Reducing the Amount of Data Downloaded

By default Mobile Client is configured to return Cover Art with a single match result, such as those returned for MusicID-Stream and MusicID-File operations.

You can decrease the amount of data your application downloads by configuring Mobile Client to not request Cover Art. See Cover Art for more details.

Improving Retrieval Performance by Using Enriched Content URLs

There may be cases when you wish to retrieve metadata, descriptors, or external identifiers as quickly as possible, while lazy-loading enriched content such as cover art in the background. To achieve this, Mobile Client features a set of APIs that allows you to extract the URL pointers to the enriched content, giving you greater control over when you load the additional content. Mobile Client provides the following APIs, each of which retrieves a different type of enriched content (cover art, artist images, album reviews and artist biographies):

- GNSearchResponse.coverArt().url()
- GNSearchResponse.contributorImage().url()
- GNSearchResponse.albumReviewUrl()
- GNSearchResponse.artistBiographyUrl()

Each API returns a string containing the URL for the enriched content. The returned value will be nil if no enriched content is available. In the case of cover art, if there is no cover art available, the URL for genre art may be returned (provided the property for genre art is set).

Enriched content URLs are temporary; therefore, the application must be configured to handle expired URLs. Gracenote currently supports content URLs for a lifespan of one hour, but this may be subject to change.

The following code examples show how to retrieve the enriched content URLs:

```
// Retrieve cover art URL
GNCoverArt *coverArt = response.coverArt;
if (coverArt !=nil) {
        NSLog(@"%@",coverArt.url);
    } else{
        NSLog(@"CoverArt is nil");
    }
// Retrieve artist image URL
GNImage *contributorGNImage = response.contributorImage;
if (contributorGNImage!=nil) {
```

```
        NSLog(@"%@",contributorGNImage.url);
    }else {
        NSLog(@"ContributorImage is nil");
    }
// Retrieve artist biography URL
NSString *artistBiographyUrl = response.artistBiographyUrl;
// Retrieve album review URL
NSString *albumReviewUrl = response.albumReviewUrl;
```

Error Handling

When the application accesses the URLs, it should handle HTTP errors gracefully. The following table lists the types of HTTP errors that might occur and the corresponding action to take:

Error	Action
HTTP 500 error	Retry fetching the URL once.
HTTP 412 error (Pre-condition failed)	The URL has expired. The application should request a new URL from the Mobile Client SDK, using GNSearchResponse.getTrackId() to get the Gracenote Identifier. The application should then use GNOperations.fetchByTrackID() before fetching the URL.
HTTP 4xx error (other 400-level errors)	Handle as if the URL has expired: request a new URL and try fetching the asset again. If that fails, the application should log an error and not attempt further retries

AlbumID Operations

As a best practice, the application should retrieve the URLs for tracks separately from GNSearch-Response after making an AlbumID request. The application should optimize and retrieve the cover art, artist image, album review, and artist biography URLs (and images and data) for one track in the album and not for each track.

Glossary

Link Data	When query results are returned by Gracenote, they can contain Link data that directly pertains to the result it is embedded in. A single piece of Link data consists of an identifier (Integer) and a source (String). The source is the provider of the identifier, such as Amazon.com. The identifier is a unique key into the provider's catalog that can be used to present the user with additional content or services. A single result can contain multiple pieces of Link data from multiple providers.

Lyric Fragment	A segment of lyrics from a track.
PCM (Pulse Coded Modulation)	Pulse Coded Modulation is the process of converting an analog in a sequence of digital numbers. The accuracy of conversion is directly affected by the sampling frequency and the bit-depth. The sampling frequency defines how often the current level of the audio signal is read and converted to a digital number. The bit-depth defines the size of the digital number. The higher the frequency and bit-depth, the more accurate the conversion.
PCM Audio	PCM data generated from an audio signal.
Result Ready Object	An object that implements a Result Ready Protocol is a Result Ready Object. An instance of a Result Ready Object must be provided when invoking a Mobile Client operation so the operation can deliver results to the application.
Result Ready Protocol	Mobile Client defines Objective-C protocols that allow Mobile Client operations to deliver results to the application. These protocols are known as Result Ready Protocols.
Single Best Match	Certain audio recognition products provided by Mobile Client return a single result; the single best match. The single best match is determined by Web Services, based on criteria specific to the application. This criteria can be refined by the application.

Troubleshooting

Sample Application Does Not Start

Problem

When running the Sample Application it does start or hangs with a blank black screen.

Solution

Ensure your Client ID is correctly provided to the GNConfig init method in the viewDidLoad method in GN_Music_SDK_iOSViewController.m.

Invalid Client Identifier Exception

Problem

When calling GNConfig.init, an exception displays with the following message:

```
invalid clientId
```

Solution

Ensure that your Client ID is correctly provided to GNConfig.init in the correct form, which is:

```
<Client ID>-<Client ID Tag>
```

IO Exception While Connecting to Web Services

Problem

The following error occurs when performing a query:

```
[Mobile:5000] webservices http status: 500: IO exception while connecting
to webservices
```

Ensure that your device and application have a working connection with the Internet.

This error can be seen due to loss of connectivity with the Internet or Internet traffic congestion on the handset.

Advanced Implementations

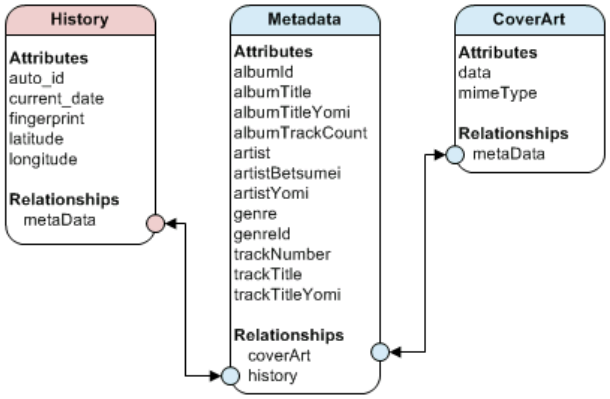
User History

The Mobile Client Sample Application includes a user history feature. It stores the results of certain operations and allows the user to view them. When viewed, the history record displays the recognized music, including Cover Art and the location of the user when the operation occurred.

The feature uses a local SQLite database to store the user history information.

Data Model

A simple data model is used to store the data as shown below.



Mobile Client Sample Application User History Database Data Model

To simplify interactions with the database, the implementation uses NSManagedObjectContext. NSManagedObjectContext presents an interface to the underlying SQLite database via objects. An object-based interface is easy to create, edit, and recall. Although the Mobile Client Sample Application uses SQLite, NSManagedObjectContext can be used to provide an object interface to various types of persistent storage.

NSManagedObjectContext allows the database object model to be completely implemented in classes. In this case the data model is implemented in:

- History.m & h
- Metadata.m & h
- CoverArt.m & h

Adding Entries

When results for an appropriate operation are received, an entry is added to the database. Only results from the following operations are stored:

- Recognizing music from the microphone
- Recognizing music from a PCM sample
- Searching by fingerprint

The process of adding an entry to the database is included in the base search operation class SearchByOperation. This allows it to be invoked in the thread that calls GNResultReady for any of the operations mentioned above. Before adding an entry, a check is made to ensure the database exists, and if it doesn't, it is created.

Each record in the database contains a unique ID. The ID is generated by incrementing the ID of the last entry added to the database. IDs are later used by the mechanism that prunes the database, keeping it within a predetermined size.

A single synchronized method is used to add an entry to the database. Synchronization is essential to guarantee the integrity of the mechanism used to generate unique IDs and the procedure used to create a database if it doesn't already exist.

Limiting Database Size

The size of the database cannot be allowed to grow indefinitely. After adding an entry to the database, its size is checked to determine if it exceeds a predefined limit of 1000 entries. If there are more than 1000 entries, the oldest entries in the database are deleted, bringing the size back to the predetermined limit.

Recalling Entries

The Mobile Client Sample Application exports a mutable copy of the entries in the database in an array. The data contained within the export can then be used to populate a UI that allows the user to navigate the entries. For large databases, a paging mechanism can be used to reduce the number of entries exported into memory at any one time.

UI Best Practices Using MusicID-Stream

Gracenote periodically conducts analysis on its MusicID-Stream product to evaluate its usage and determine if there are ways we can make it even better. Part of this analysis is determining why some MusicID-Stream recognition queries do not find a match.

Consistently Gracenote finds that the majority of failing queries contain an audio sample of silence, talking, humming, singing, whistling or live music. These queries fail because the Gracenote MusicID-Stream service can only match commercially released music.

Such queries are shown to usually originate from applications that do not provide good end user instructions on how to correctly use the MusicID-Stream service. Therefore Gracenote recommends that application developers consider incorporating end user instructions into their applications from the beginning of the design phase. This section describes the Gracenote recommendations for instructing end users on how to use the MusicID-Stream service in order to maximize recognition rates and have a more satisfied user base.

This section is specifically targeted to applications running on a user's cellular handset, tablet computer, or similar portable device, although end user instructions should be considered for all applications using MusicID-Stream. Not all recommendations listed here are feasible for every application. Consider them options for improving your application and the experience of your end users.

Clear and Accessible

All instructions provided to the user should be easy to understand and accessible at any time. For example:

- Use pictures instead of text
- Provide a section in the device user manual (where applicable)
- Provide a help section within the application
- Include interactive instructions embedded within the flow of the application. For example, prompt the user to hold the device to the audio source.

Rotating Help Message upon Failed Recognition

When a recognition attempt fails, display a help message with a hint or tip on how to best use the MusicID-Stream service; a concise, useful tip can persuade a user to try again. Have a selection of help messages available; show one per failed recognition attempt but rotate which message is displayed.

Allow the User to Provide Feedback

When a recognition attempt fails, allow the user to submit a hint with information about what they are looking for. Based on the response, the application could return a targeted help message about the correct use of MusicID-Stream.

Audio Recording Animation

While the device is recording an audio sample, display a simple image or animation that explains how to correctly use MusicID-Stream.

Audio Recording Progress Indicator or Countdown

Use a progress bar or countdown to indicate how long the application will be recording. The user can use this information to assist in collecting an audio sample that can be successfully recognized.

Audio Recording Completed Cues

Some failing MusicID-Stream recognitions are caused by insufficient or incomplete recording of the song. To assist the user in assessing if recording is complete, visual, audible and tangible cues can be used, such as:

- Display a message (only useful if the user is looking at the display)
- Sound a tone or tune (not useful in load environments)
- Vibrate the handset (always useful)

Use Street Sign-like Images

Street sign images are universal and easily recognizable. These can be used to provide clear instructions about where and how to use and not to use the MusicID-Stream service.

Demo Animation

Provide a small, simple animation that communicates how to use the application. Make this animation accessible at all times from the Help section.

If requested (and if the application is so entitled), Mobile Client can also provide cover art and Link identifiers with the recognition results; see Retrieving Cover Art and Retrieving Link Data, below, for further information.