GNSDK for Mobile (iOS) Developers Guide

Release 1.1.2.2866

Published: 10/29/20149:42 AM

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

Preface

Preface ii

Start Here - Doc Roadmap x

Doc Set Summary x

API Reference Documentation x

Sample Applications x

Concepts 1

Introduction 1

About Gracenote 1

What is Gracenote SDK? 1

GNSDK for Mobile and Gracenote Products 1

Modules Overview 2

General GNSDK for Mobile Modules 2

Product-Specific Modules 2

Gracenote MetaData 4

Gracenote Media Elements 4

Music Terminology 4

Genre and Other List-Dependent Values 4

Genres and List Hierarchies 5

Simplified and Detailed Hierarchical Groups 6

Core and Enriched Metadata 6

Mood and Tempo (Sonic Attributes) 7

Classical Music Metadata 7

Music Modules 9

Music Module Overview 9

MusicID Overview 9

Text-Based Recognition 10

MusicID-Stream 10

Discovery Features 11

Playlists 11

© 2000 to present. Gracenote, Inc. All rights reserved.

Collection Summaries 11
More Like This 11
.Playlist Requirements and Recommendations 11
Simplified Playlist Implementation 12
Playlist Content Requirements 12
Playlist Storage Recommendations 12
Playlist Resource Requirements 12
Playlist Level Equivalency for Hierarchical Attributes 13
Key Playlist Components 13
Media metadata: Metadata of the media items (or files) 13
Attributes: Characteristics of a media item, such as Mood or Tempo 13
Unique Identifiers 14
Collection Summary 14
Storage 14
Seed 14
Playlist generation 14
MoodGrid Overview *15*
Platform Development Guidelines **17**
Supported Platforms and System Requirements **17**
Modules in the GNSDK Package *17*
GNSDK for Mobile Modules 17
Memory Usage *17*
Memory Usage Guidelines 18
RAM Requirements 18
Playlist Memory Usage 19
Performance *19*
CD Lookup Performance 19
Text Lookup Performance 20
TrackID (Fingerprint) Performance 20
I/O Performance 20
iOS **21**

© 2000 to present. Gracenote, Inc. All rights reserved.

Deployment *21*
Creating an iPhone Development Environment 21
Advanced Implementations *21*
User History 21
Adding Entries 22
Limiting Database Size 22
Recalling Entries 22
Mobile Getting Started **23**
Getting Started with Mobile iOS **23**
Introduction *23*
Fingerprint and Metadata Samples *23*
Setting Up the Development Environment *23*
Requirements 23
Technical Requirements 24
Framework Size 24
Apple Framework Dependencies 24
Linker Flags 24
Installing the iOS SDK *24*
Building and Running the Sample Applications *25*
Testing the Sample Applications 25
Implementing Applications (All Platforms) **27**
About the Implementing Applications Documentation **27**
Basic Application Steps **27**
Setup and Initialization **28**
Authorizing a GNSDK Application *28*
Client ID 29
License File 29
Including Header Files *29*
Instantiating a GNSDK Manager Object *29*
Specifying the License File 29
Instantiating a User Object *30*

© 2000 to present. Gracenote, Inc. All rights reserved.

Saving the User Object to Persistent Storage 31
Loading a Locale **32**
Locale-Dependent Data *33*
Default Regions and Descriptors *33*
Locale Groups *34*
Multi-Threaded Access 34
Updating Locales and Lists 34
Update Notification 34
Locale Behavior *35*
Best Practices *36*
Using Gracenote SDK Logging **37**
Enabling GNSDK Logging *38*
Implementing Callback Logging *39*
Enabling and Using GNSDK Storage **39**
Enabling a Provider for GNSDK Storage *40*
GNSDK Stores *40*
GNSDK Databases *41*
Setting GNSDK Storage Folder Locations *41*
Getting Local Lookup Database Information *42*
Image Information 42
Database Versions 42
Getting Available Locales 42
Setting Online Cache Expiration *42*
Managing Online Cache Size and Memory *42*
Setting Local and Online Lookup Modes **43**
Supported Lookup Modes *43*
Default Lookup Mode *44*
Setting the Lookup Mode for a User or Specific Query *44*
Using Both Local and Online Lookup Modes *44*
Using a Local Fingerprint Database **44**
Downloading and Ingesting Bundles *45*

© 2000 to present. Gracenote, Inc. All rights reserved.

Designating a Storage Provider *48*
Identifying Music **48**
Identifying Music Using a CD-TOC, Text, or Fingerprints (MusicID) *49*
MusicID Queries 49
Options for MusicID Queries 49
Identifying Music Using a CD TOC 50
Identifying Music Using Text 51
Identifying MusCic Using Fingerprints 52
PCM Audio Format Recommendations 52
MusicID Fingerprinting 52
Identifying Audio Files (MusicID-File) *54*
Implementing an Events Delegate 55
Adding Audio Files for Identification 58
Setting Audio File Identification 59
MusicID-File Fingerprinting 61
Setting Options for MusicID-File Queries 64
Making a MusicID-File Query 64
Options When Making Query Call 64
Identifying Streaming Audio (MusicID-Stream) *65*
Setting Options for Streaming Audio Queries 68
Music-ID Stream Code Samples 68
Processing Returned Metadata Results **74**
Needs Decision *74*
Full and Partial Metadata Results *75*
Locale-Dependent Data *76*
Accessing Enriched Content **76**
Setting the Query Option for Enriched Content *76*
Processing Enriched Content *76*
Discovery Features **80**
Generating a Playlist *80*
Creating a Collection Summary 80

© 2000 to present. Gracenote, Inc. All rights reserved.

Populating a Collection Summary 81
Retrieving Playlist Attributes in Queries 82
How Playlist Gathers Data 82
Working with Local Storage 82
Generating a Playlist Using More Like This 83
Generating a Playlist Using PDL (Playlist Description Language) 84
Accessing Playlist Results 85
Working with Multiple Collection Summaries 88
Join Performance and Best Practices 88
Synchronizing Collection Summaries 89
Iterating the Physical Media 89
Processing the Collection 89
Playlist PDL Specification *89*
PDL Syntax 89
Keywords 90
Example: Keyword Syntax 91
Operators 91
Literals 91
Attributes 92
Functions 92
PDL Statements 92
Attribute Implementation <att_imp> 93
Expression <expr> 93
Score <score> 93
Example: PDL Statements 93
Implementing Moodgrid *94*
Prerequisites 95
Enumerating Data Sources using MoodGrid Providers 95
Creating and Populating a MoodGrid Presentation 95
Iterating Through a MoodGrid Presentation 96
Filtering MoodGrid Results 98

© 2000 to present. Gracenote, Inc. All rights reserved.

Implementing Rhythm *98*
Creating a Rhythm Query 98
Setting Rhythm Query Options 99
Generating Recommendations 100
Creating a Radio Station and Playlist 100
Providing Feedback 101
Tuning a Radio Station 101
Saving and Retrieving Radio Stations 102
Best Practices and Design Requirements **102**
Image Best Practices *102*
Image Resizing Guidelines 102
UI Best Practices for Audio Stream Recognition *103*
Provide Clear and Accessible Instructions 104
Provide a Demo Animation 104
Display a Progress Indicator During Recognition 104
Use Animations During Recognition 104
Using Vibration, Tone, or Both to Indicate Recognition Complete 104
Display Help Messages for Failed Recognitions 104
Allow the User to Provide Feedback 105
Data Models **106**
API Reference Documentation **107**
Playlist PDL Specification **107**
PDL Syntax **107**
Keywords *107*
Example: Keyword Syntax 109
Operators *109*
Literals *109*
Attributes *109*
Functions *110*
PDL Statements *110*
Attribute Implementation <att_imp> 110

© 2000 to present. Gracenote, Inc. All rights reserved.

Expression <expr> 111
Score <score> 111
Example: PDL Statements **111**
GNSDK Glossary **112**

© 2000 to present. Gracenote, Inc. All rights reserved.

Start Here - Doc Roadmap

GNSDK for Mobile (iOS) provides a broad range of documentation and sample code to help you develop GNSDK applications. The documentation is in two formats:

- HTML5 Help
- PDF

In general, each of these contain the same content. The main difference is that the PDF documentation does not contain API Reference content because of its size and complexity. API Reference documentation is generated from the GNSDK source files and provided in HTML format accessible from the HTML5 Help system.

Doc Set Summary

GNSDK for Mobile (iOS) has the following doc set:

Document	Description
GNSDK for Mobile (iOS) Help	The most comprehensive format of GNSDK for Mobile (iOS) documentation. Includes API Reference and Developers Guide content.
GNSDK for Mobile (iOS) Developers Guide	Detailed information about developing GNSDK for Mobile (iOS) applications in a printable format. No API Reference content.
GNSDK for Mobile (iOS) Release Notes (*PDF only*)	Summary of what's new and changed in this release.

API Reference Documentation

You can develop GNSDK for Mobile applications using several popular object-oriented languages. The following table describes these APIs and where to find the corresponding API Reference documentation:

API Name	Description	Location in Package
Objective-C API Reference	API descriptions of the GNSDK for Mobile Objective-C interface.	docs/Content/api_ref_objc/html/index.html

Sample Applications

Code Samples	Description	Location in Package

© 2000 to present. Gracenote, Inc. All rights reserved.

Objective-C Sample applications	Applications you can compile and run that demonstrate specific GNSDK for Mobile (iOS)features.	samples_objc

© 2000 to present. Gracenote, Inc. All rights reserved.

Concepts

Introduction

About Gracenote

A pioneer in the digital media industry, Gracenote combines information, technology, services, and applications to create ingenious entertainment solutions for the global market.

From media management, enrichment, and discovery products to content identification technologies, Gracenote allows providers of digital media products and the content community to make their offerings more powerful and intuitive, enabling superior consumer experiences. Gracenote solutions integrate the broadest, deepest, and highest quality global metadata and enriched content with an infrastructure that services billions of searches a month from thousands of products used by hundreds of millions of consumers.

Gracenote customers include the biggest names in the consumer electronics, mobile, automotive, software, and Internet industries. The company's partners in the entertainment community include major music publishers and labels, prominent independents, and movie studios.

Gracenote technologies are used by leading online media services, such as Apple iTunes®, Pandora®, and Sony Music Unlimited, and by leading consumer electronics manufacturers, such as Pioneer, Philips, and Sony, and by nearly all OEMs and Tier 1s in the automotive space, such as GM, VW, Nissan, Toyota, Hyundai, Ford, BMW, Mercedes Benz, Panasonic, and Harman Becker.

For more information about Gracenote, please visit: www.gracenote.com.

What is Gracenote SDK?

Gracenote SDK (GNSDK) is a platform that delivers Gracenote technologies to devices, desktop applications, web sites, and backend servers. GNSDK enables easy integration of Gracenote technologies into customer applications and infrastructure—helping developers add critical value to digital media products, while retaining the flexibility to fulfill almost any customer need.

GNSDK is designed to meet the ever-growing demand for scalable and robust solutions that operate smoothly in high-traffic server and multithreaded environments. GNSDK is also lightweight—made to run in desktop applications and even to support popular portable devices.

GNSDK for Mobile and Gracenote Products

GNSDK for Mobile provides convenient access to Gracenote Services (Gracenote's suite of advanced media recognition, enriched content, and discovery services paired with robust infrastructure), enabling easy integration of powerful Gracenote products into customer applications and devices. Gracenote products provided by GNSDK for Mobile are summarized in the table below.

© 2000 to present. Gracenote, Inc. All rights reserved.

Gracenote Products	GNSDK for Mobile Modules	Module Description
MusicID TM	MusicID, MusicID-File	MusicID (CD TOC, Text, Stream and File Fingerprint); MusicID-File (LibraryID, AlbumID and TrackID Fingerprint Identification) Enables MusicID recognition for identifying CDs, digital music files, and streaming audio and delivers relevant metadata such as track titles, artist names, album names, and genres. Also provides library organization and direct lookup features.
Playlist	Playlist, MoodGrid	Generates playlists based on music descriptors. Includes More Like This™ functionality to generate a playlist similar to the currently playing track. Also supports creation of MoodGrids to visually group content based on mood metadata.

Modules Overview

GNSDK for Mobile consists of several modules that support specific Gracenote products. The principal module required for all applications is the GNSDK Manager. Others are optional, depending on the functionality of the applications you develop and the products you license from Gracenote.

General GNSDK for Mobile Modules

GNSDK Manager: Required. GNSDK Manager provides core functionality necessary to all Gracenote modules.

Contact your Gracenote Global Services & Support representative for more information.

SQLite: . The SQLite module provides a local storage solution for GNSDK for Mobile using the SQLite database engine. This module is primarily used to manage a local cache of queries and content that the GNSDK for Mobile modules make to Gracenote Service. SQLite is a software module that implements a self-contained, server-less, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain. Information on SQLite can be found at http://www.sqlite.org.

Product-Specific Modules

MusicID: Provides CD TOC, text, and fingerprint identification for music.

MusicID-File: Provides file-based music identification using LibraryID, AlbumID, and TrackID processing functionality.

MusicID-Stream: Designed specifically to recognize music delivered as a continuous stream. Use this library in applications that need to recognize streaming music in real time and on-demand.

© 2000 to present. Gracenote, Inc. All rights reserved.

Playlist: Provides functionality to generate and manage playlists and playlist collections. Also supports creation of MoodGrids to visually group content based on mood metadata.

© 2000 to present. Gracenote, Inc. All rights reserved.

Gracenote MetaData

Gracenote Media Elements

Gracenote Media Elements are the software representations of real-world things like CDs, Albums, Tracks, Contributors, and so on. The following is a partial list of the higher-level media elements represented in GNSDK for Mobile:

Music

- Music CD
- Album
- Track
- Artist
- Contributor

Music Terminology

The following terms are used throughout the music documentation. For a detailed list of GNSDK terms and definitions, see the Glossary.

- **Album** - A collection of audio recordings, typically of songs or instrumentals.
- **Audio Work** - A collection of classical music recordings.
- **Track** - A song or instrumental recording.
- **Artist** - The person or group primarily responsible for creating the Album or Track.
- **Contributor** - A person that participated in the creation of the Album or Track.

Genre and Other List-Dependent Values

GNSDK for Mobile uses list structures to store strings and other information that do not directly appear in results returned from Gracenote Service. Lists generally contain information such as localized strings and region-specific information. Each list is contained in a corresponding List Type.

Some Gracenote metadata is grouped into hierarchical lists. Some of the more common examples of list-based metadata include genre, artist origin, artist era, and artist type. Other list-based metadata includes mood, tempo, roles, and others.

Lists-based values can vary depending on the locale being used for an application. That is, some values will be different depending on the chosen locale of the application. Therefore, these kinds of list-based metadata values are called *locale-dependent.*

© 2000 to present. Gracenote, Inc. All rights reserved.

Genres and List Hierarchies

One of the most commonly used kinds of metadata are genres. A genre is a categorization of a musical composition characterized by a particular style. The list system enables genre classification and hierarchical navigation of a music collection. The Genre list is very granular for two reasons:

- To ensure that music categories from different countries are represented and that albums from around the world can be properly classified and represented to users based on the user's geographic location and cultural preferences.
- To relate different regionally-specific music classifications to one another, to provide a consistent user experience in all parts of the world when creating playlists with songs that are similar to each other.

The Gracenote Genre System contains more than 2200 genres from around the world. To make this list easier to manage and give more display options for client applications, the Gracenote Genre System groups these genres into a relationship hierarchy. Most hierarchies consists of three levels: level-1. level-2, and level-3.

- Level-1
  - Level-2
    - Level-3

For example, the partial genre list below shows two, level-1 genres: Alternative & Punk and Rock.

Each of these genres has two, level-2 genres. For Rock, the level-2 genres shown are Heavy Metal and 50's Rock. Each level-2 genre has three level-3 genres. For 50's Rock, these are Doo Wop, Rockabilly, and Early Rock and Roll. This whole list represents 18 genres.

- Alternative & Punk
  - Alternative
    - Nu-Metal
    - Rap Metal
    - Alternative Rock
  - Punk
    - Classic U.K. Punk
    - Classic U.S. Punk
    - Other Classic Punk
- Rock
  - Heavy Metal
    - Grindcore
    - Black Metal
    - Death Metal
  - 50's Rock
    - Doo Wop
    - Rockabilly
    - Early Rock & Roll

Other category lists include: origin, era, artist type, tempo, and mood.

© 2000 to present. Gracenote, Inc. All rights reserved.

Simplified and Detailed Hierarchical Groups

In addition to hierarchical levels for some metadata, the Gracenote Genre System provides two general kinds of hierarchical groups (also called list hierarchies). These groups are called *Simplified* and *Detailed*.

You can choose which hierarchy group to use in your application for any of the locale/list-dependent values. Your choice depends on how granular you want the metadata to be.

The Simplified group retrieves the coarsest (largest) grain, and the Detailed group retrieves the finest (smallest) grain, as shown below.

Example of a Simplified Genre Hierarchy Group

- Level-1: 10 genres
  - Level-2: 75 genres
    - Level-3: 500 genres

Example of a Detailed Genre Hierarchy Group

- Level-1: 25 genres
  - Level-2: 250 genres
    - Level-3: 800 genres

Core and Enriched Metadata

All Gracenote customers can access core metadata from Gracenote for the products they license. Optionally, customers can retrieve additional metadata, known as *enriched metadata*, by purchasing additional metadata entitlements.

The following diagram shows the core and enriched metadata available for Music.

© 2000 to present. Gracenote, Inc. All rights reserved.

Mood and Tempo (Sonic Attributes)

Gracenote provides two metadata fields that describe the sonic attributes of an audio track. These fields, mood and tempo, are track-level descriptors that capture the unique characteristics of a specific recording.

Mood is a perceptual descriptor of a piece of music, using emotional terminology that a typical listener might use to describe the audio track. Mood helps power Gracenote Playlist.

Tempo is a description of the overall perceived speed or pace of the music. Gracenote mood and tempo descriptor systems include hierarchical categories of increasing granularity, from very broad parent categories to more specific child categories.

NOTE: Tempo metadata is available online-only.

Classical Music Metadata

This topic discusses Gracenote classical music support, Three-Line-Solution (TLS), and GNSDK for Mobile's support for accessing classical music metadata from Gracenote Service. Gracenote TLS provides the four basic classical music metadata components—composer, album name, artist name, and track name—in a standard three-field media display for albums, artists, and tracks, as follows:

Classical Three-Line Display: Metadata Mapping

Field	Three-line Display
Track	[Composer Short Name]: [Work Title] In {Key}, {Opus}, {Cat#}, {“Nickname”} – [Movement#]. [M. Name]
Artist	{Soloist(s)}, {Conductor}; {Ensemble}, {Choral Ensemble}
Album	As printed on the spine

Classical Three-Line Display: Metadata Mapping Example

Field	Three-line Display
Track	Vivaldi: The Four Seasons, Op. 8/1, “Spring” – 1. Allegro
Artist	Joseph Silverstein, Seiji Ozawa; Boston Symphony Orchestra
Album	Vivaldi: The Four Seasons

- Composer: The composer(s) that are featured on an album are listed by their last name in the album title (where applicable). In the examples noted below, the composer is Beethoven.
- Album Name: In most cases, a classical album’s title is comprised of the composer(s) and work(s) that are featured on the product, which yields a single entity in the album name display. However, for albums that have formal titles (unlike composers and works), the title is listed as it is on the product.
- General title example: Beethoven: Violin Concerto
- Formal title example: The Best Of Baroque Music

© 2000 to present. Gracenote, Inc. All rights reserved.

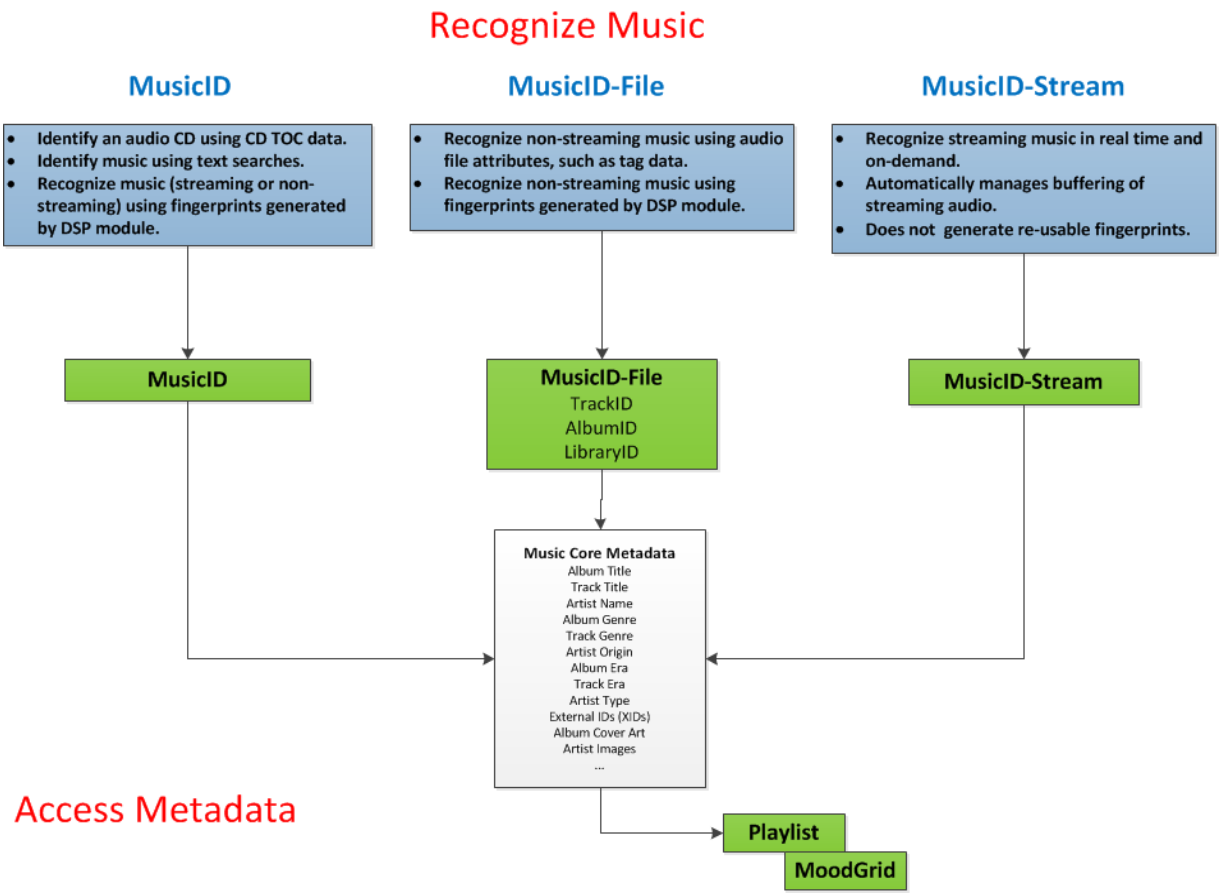
- Artist Name: A consistent format is used for listing a recording artist(s)—by soloist(s), conductor, and ensemble(s)—which yields a single entity in the artist name display. For example: Hilary Hahn; David Zinman: Baltimore Symphony Orchestra
- Track Name: A consistent format is used for listing a track title—composer, work title, and (where applicable) movement title—which yields a single entity in the track name display. For example: Beethoven: Violin Concerto In D, Op. 61 – 1. Allegro Ma Non Troppo

© 2000 to present. Gracenote, Inc. All rights reserved.

Music Modules

Music Module Overview

The following diagram summarizes the kinds of identification queries each Music module supports.



MusicID Overview

MusicID allows application developers to deliver a compelling digital entertainment experience by giving users tools to manage and enjoy music collections on media devices, including desktop and mobile devices. MusicID is the most comprehensive identification solution in the industry with the ability to recognize, categorize and organize any music source, be it CDs, digital files, or audio streams. MusicID also seamlessly integrates with Gracenote's suite of products and provides the foundation for advanced services such as enriched content and linking to commerce.

Media recognition using MusicID makes it possible for applications to access a variety of rich data available from Gracenote. After media has been recognized, applications can request and utilize:

© 2000 to present. Gracenote, Inc. All rights reserved.

- Album, track, and artist names
- Genre, origin, era and type descriptors

GNSDK for Mobile accepts the following types of inputs for music recognition:

- CD TOCs
- File fingerprints
- Stream fingerprints
- Text input of album and track titles, album and track artist names, and composer names
- Media element identifiers
- Audio file and folder information (for advanced music recognition)

Text-Based Recognition

You can identify music by using a lookup based on text strings. The text strings can be extracted from an audio track's file path name and from text data embedded within the file, such as mp3 tags. You can provide the following types of input strings:

1. Album title
2. Track title
3. Album artist
4. Track artist
5. Track composer

Text-based lookup attempts to match these attributes with known albums, artists, and composers. The text lookup first tries to match an album. If that is not possible, it next tries to match an artist. If that does not succeed, a composer match is tried. Adding as many input strings as possible to the lookup improves the results.

If a query handle populated with text inputs is passed to gnsdk_musicid_find_matches(), then best-fit objects will be returned. In this instance, you might get back album matches or contributor matches or both. Album matches are generally ranked higher than contributor matches

---

MusicID-Stream

You can use MusicID-Stream to recognize music delivered as a continuous stream. Specifically, MusicID-Stream performs these functions:

- Recognizing streaming music in real time and on-demand .
- Automatically manages buffering of streaming audio.
- Continuously identifies the audio stream when initiated until it generates a response.

After establishing an audio stream, the application can trigger an identification query at any time. For example, this action could be triggered on-demand by an end user pressing an "Identify" button provided by the application UI.

© 2000 to present. Gracenote, Inc. All rights reserved.

The identification process identifies the buffered audio. Up to seven seconds of the most recent audio is buffered. If there is not enough audio buffered for identification, MusicID-Stream waits until enough audio is received. The identification process spawns a thread and completes asynchronously.

The application can identify audio using Gracenote Service online database or a local database. The default behavior attempts a local database match. If this fails, the system attempts an online database match.

Discovery Features

Playlists

Playlist provides advanced playlist generation enabling a variety of intuitive music navigation methods. Using Playlist, applications can create sets of related media from larger collections—enabling valuable features such as More Like This™ and custom playlists—that help users easily find the music they want.

Playlist functionality can be applied to both local and online user collections. Playlist is designed for both performance and flexibility—utilizing lightweight data and extensible features.

Playlist builds on the advanced recognition technologies and rich metadata provided by Gracenote through GNSDK for Mobile to generate highly relevant playlists.

Collection Summaries

Collection summaries store attribute data to support all media in a candidate set and are the basis for playlist generation. Collection summaries are designed for minimal memory utilization and rapid consumption, making them easily applicable to local and server/cloud-based application environments.

Playlists are generated using the attributes stored in the active collection summary. Collection summaries must, therefore, be refreshed whenever media in the candidate set or attribute implementations are modified.

Playlist supports multiple collection summaries, enabling both single and multi-user applications.

More Like This

More Like This is a powerful and popular navigation feature made possible by Gracenote and GNSDK for Mobile Playlist. Using More Like This, applications can automatically create a playlist of music that is similar to user-supplied seed music. More Like This is commonly applied to an application's currently playing track to provide users with a quick and intuitive means of navigating through their music collection.

Gracenote recommends using More Like This to quickly implement a powerful music navigation solution. Functionality is provided via a dedicated API to further simplify integration. If you need to create custom playlists, you can use the Playlist Definition Language.

.Playlist Requirements and Recommendations

This topic discusses requirements and recommendations for your Playlist implementation.

© 2000 to present. Gracenote, Inc. All rights reserved.

Simplified Playlist Implementation

Gracenote recommends streamlining your implementation by using the provided More Like This function, gnsdk_playlist_generate_morelikethis(). It uses the More Like This algorithm to generate optimal playlist results and eliminates the need to create and validate Playlist Definition Language statements.

Playlist Content Requirements

Implementing Playlist has these general requirements:

- The application integrates with GNSDK for Mobile's MusicID or MusicID-File (or both) to recognize music media and create valid GDOs.
- The application uses valid unique identifiers. A unique identifier must be a valid UTF-8 string of unique media identifier data. For more information, see "Unique Identifiers" on page 14.

Unique identifiers are essential to the Playlist generation process. The functionality cannot reference a media item if its identifier is missing or invalid.

Playlist Storage Recommendations

GNSDK for Mobile provides the SQLite module for applications that may need a storage solution for collections. You can dynamically create a collection and release it when you are finished with it. If you choose this solution, you must store the GDOs or recognize the music at the time of creating the collection.

Your application can also store the collection using the serialization and deserialization functions.

Playlist Resource Requirements

The following table lists resource requirements for Playlist's two implementation scenarios:

Use Case	Typical Scenario	Number of Collection Summaries	Application Provides Collection Summary to Playlist	Required Computing Resources
Single user	Desktop user Mobile device user	Generally only one	Once, normally at start-up	Minimal-to-average, especially as data is ingested only once or infrequently

© 2000 to present. Gracenote, Inc. All rights reserved.

Use Case	Typical Scenario	Number of Collection Summaries	Application Provides Collection Summary to Playlist	Required Computing Resources
Multiple users	Playlist server Playlist - in-the-cloud system	Multiple; requires a unique collection summary for each user who can access the system	Dynamically and multiple times; typically loaded with the playlist criteria at the moment before playlist generation	Requires more computing resources to ensure an optimal user experience

Playlist Level Equivalency for Hierarchical Attributes

Gracenote maintains certain attribute descriptors, such as Genre, Era, Mood, and Tempo, in multi-level hierarchies. For a descriptions of the hierachies, see "Mood and Tempo (Sonic Attributes)" on page 7. As such, Playlist performs certain behaviors when evaluating tracks using hierarchical attribute criteria.

Track attributes are typically evaluated at their equivalent hierarchy list-level. For example, Rock is a Level 1 genre. When evaluating candidate tracks for a similar genre, Playlist analyzes a track's Level 1 genre.

However, Seeds contain the most granular-level attribute. When using a SEED, Playlist analyzes tracks at the respective equivalent level as is contained in the Seed, either Level 2 or Level 3.

Key Playlist Components

Playlist operates on several key components. The GNSDK for Mobile Playlist module provides functions to implement and manage the following key components within your application.

Media metadata: Metadata of the media items (or files)

The media may be on MP3s on a device, or a virtual collection hosted on a server. Each media item must have a unique identifier, which is application-defined.

Playlist requires recognition results from GNSDK for Mobile for operation, and consequently must be implemented with one or both of GNSDK for Mobile's music identification modules, MusicID and MusicID-File.

Attributes: Characteristics of a media item, such as Mood or Tempo

Attributes are Gracenote-delivered string data that an application can display; for example, the Mood attribute Soulful Blues.

When doing recognition queries, if the results will be used with Playlist, set the ‘enable playlist’ query option to ensure proper data is retrieved for the result (GNSDK_MUSICID_OPTION_ENABLE_PLAYLIST or GNSDK_MUSICIDFILE_OPTION_ENABLE_PLAYLIST).

© 2000 to present. Gracenote, Inc. All rights reserved.

Unique Identifiers

When adding media to a collection summary, an application provides a unique identifier and a GDO, which contains the metadata for the identifier. The identifier is a value that allows the application to identify the physical media being referenced. The identifier is not interpreted by Playlist; it is only returned to the application in Playlist results. An identifier is generally application-dependent. For example, a desktop application typically uses a full path to a file name for an identifier, while an online application typically uses a database key. The media GDO should contain relevant metadata for the media referenced by the identifier. In most cases the GDO comes from a recognition event for the respective media (such as from MusicID). Playlist will take whatever metadata is relevant for playlist generation from the given GDO. For best results, Gracenote recommends giving Album Type GDOs that have matched tracks to this function; Track Type GDOs also work well. Other GDOs are supported, but most other types lack information for good Playlist generation.

Collection Summary

A collection summary contains the distilled information GNSDK for Mobile Playlist uses to generate playlists for a set of media.

Collection summaries must be created and populated before Playlist can use them for playlist generation. Populating collection summaries involves passing a GDO from another GNSDK for Mobile identification event (for example, from MusicID) along with a unique identifier to Playlist. Collection Summaries can be stored so they do not need to be reconstructed before every use.

Storage

The application can store and manage collection summaries in local storage.

Seed

A seed is the GDO of a media item in the collection summary used as input criteria for playlist generation. A seed is used to generate More Like This results, or in custom PDL statements. For example, the seed could be the GDO of a particular track that you'd like to use to generate More Like This results.

Playlist generation

GNSDK for Mobile provides two Playlist generation functions: a general function for generating any playlist, gnsdk_playlist_generate_playlist(), and a specific function for generating a playlist that uses the Gracenote More Like This algorithm, gnsdk_playlist_generate_morelikethis().

NOTE: Gracenote recommends streamlining your Playlist implementation by using the provided More Like This™ function, gnsdk_playlist_generate_morelikethis(), which uses the More Like This algorithm to generate optimal playlist results and eliminates the need to create and validate PDL statements.

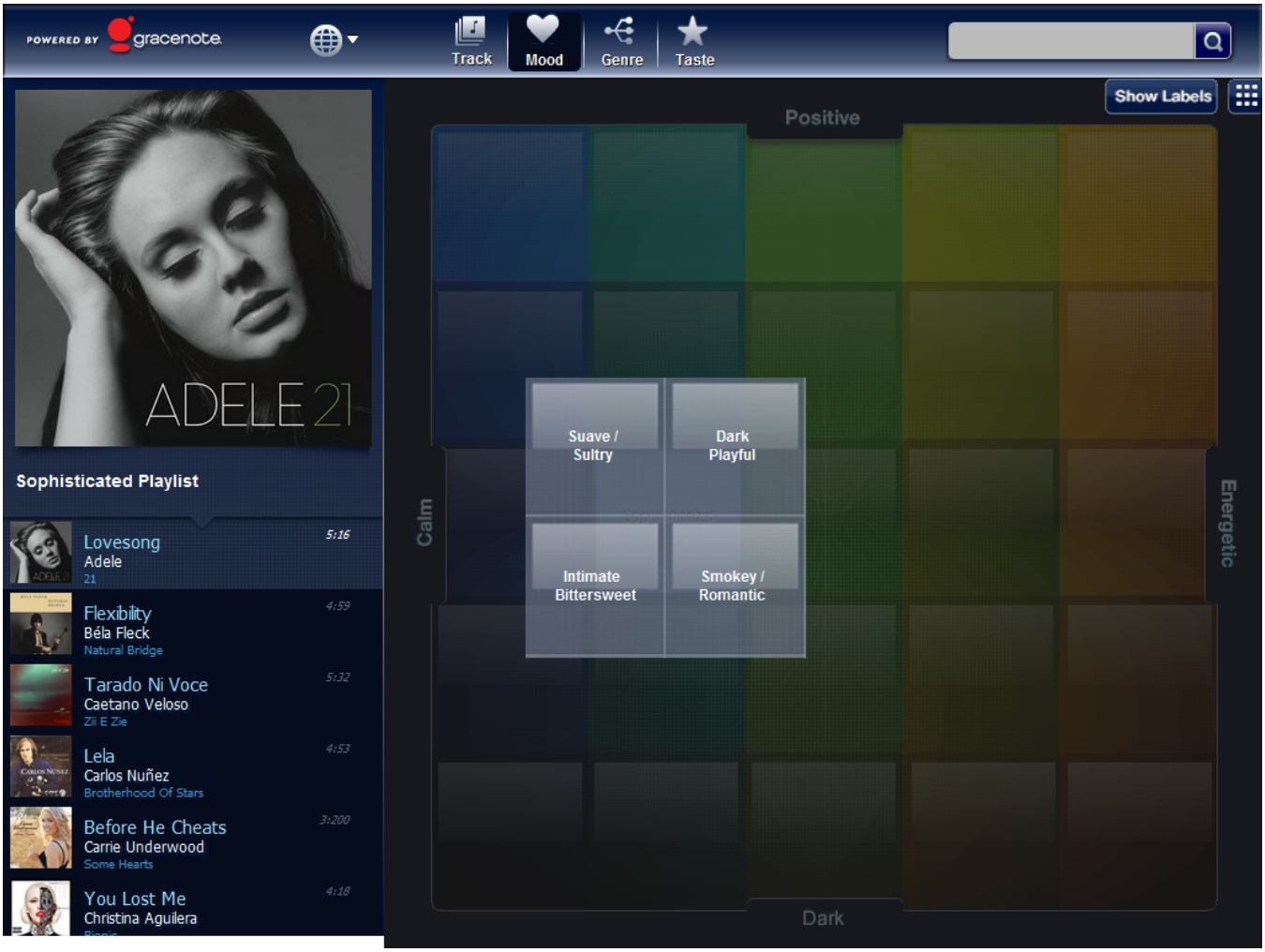
© 2000 to present. Gracenote, Inc. All rights reserved.

MoodGrid Overview

The MoodGrid library allows applications to generate playlists and user interfaces based on Gracenote Mood descriptors. MoodGrid provides Mood descriptors to the application in a two-dimensional grid that represents varying degrees of moods across each axis. One axis represents energy (calm to energetic) and the other axis represents valence (dark to positive). When the user selects a mood from the grid, the application provides a playlist of music that corresponds to the selected mood. Additional filtering support is provided for genre, origin, and era music attributes.

Mood Descriptors are delivered with track metadata from queries to MusicID, MusicID-File, or MusicID-Stream services.

The MoodGrid data representation is a 5x5 or 10x10 grid. Each element of the grid corresponds to a Gracenote Mood category ID (Level 1 for 5x5 and Level 2 for 10x10), which can be used to create a list of playable songs in the user's collection characterized by that mood. Each Level 1 Mood maps to four, more granular, Level 2 Moods. For example, a Level 1 Mood might be "Romantic", and the corresponding Level 2 Moods might be "Suave/Sultry", "Dark/Playful", "Intimate/Bittersweet", and "Smokey/Romantic", providing a greater level of detail within the "Romantic" Mood.



© 2000 to present. Gracenote, Inc. All rights reserved.

Note: The pictured implementation of MoodGrid is centered upon the collection and arrangement of track Mood descriptors across the dimensions of energy (calm to energetic) and valence (dark to positive). The actual layout and navigation of these moods is entirely dependent upon the implementation of the application's user interface. Gracenote's MoodGrid does not necessarily need to be presented in the square representation demonstrated above.

© 2000 to present. Gracenote, Inc. All rights reserved.

Platform Development Guidelines

Supported Platforms and System Requirements

GNSDK for Mobile provides multi-threaded, thread-safe technology for the following common platforms.

Platform	Architecture	Bit	S/S	OS	Compiler/Note
iOS	ARMV7, ARMV7s, x86 Emulator	32-bit	Static only	iOS 6.0 and higher	Xcode 5.0 and higher
Windows Phone 8, Windows RT	VSIX	32-bit		Windows 8 or later	Microsoft Visual Studio 2012 Update 4 or later. GNSDK libraries are provided as a VSIX installation package.

NOTE: All platforms above go through QA testing, however, only select platforms go through full regression testing.

Modules in the GNSDK Package

GNSDK provides the following modules for application development. For more information about these modules, search this documentation or refer to the table of contents.

GNSDK for Mobile Modules

- DSP
- LOOKUP_LOCALSTREAM
- MANAGER
- MOODGRID
- MUSICID
- MUSICID_FILE
- MUSICID_STREAM
- PLAYLIST
- STORAGE_SQLITE

Memory Usage

The memory usage of GNSDK for Mobile depends on a number of different factors, including the use of multiple threads, type of recognition, size of a user's collection, number of simultaneous devices connected, metadata requested, etc. Typical integrations can use anywhere from 5 MB to 30 MB depending on what use cases the application is addressing. There is no "magic number" to describe the memory usage requirements of GNSDK for Mobile.

© 2000 to present. Gracenote, Inc. All rights reserved.

Memory Usage Guidelines

The following guidelines for RAM are based on measurements of both local and online lookups in a single-threaded environment on the BeagleBoard xM embedded platform. Use of multiple threads, such as with multiple online lookups, can theoretically result in increased maximum memory usage. However, this would require highly unlikely alignment of threads.

Beagle Board xM	
CPU Type	ARM Cortex A8
CPU Speed	1 GHz
CPU Endianness	Little
Storage Size	512 MB
Storage Type	NAND Flash
RAM size	512 MB
RAM type	LPDDR
OS	Angstrom Linux

RAM Requirements

GNSDK for Mobile may utilize more memory than indicated below. Gracenote always recommends that customers measure heap usage on their device running their application code and utilizing Gracenote's memory usage callback functionality.

Measured RAM Requirements describe the memory usage using the hardware, software and test sets outlined. The measured RAM requirements are for peak usages and will be smaller the majority of time. Recommended RAM Requirements account for future Gracenote feature enhancements.

Feature Set	**Measured RAM Requirements***	**Recommended RAM Requirements***
MusicID (CD, Text) Music Enrichment (Cover Art)	3 MB	8 MB
MusicID (CD, Text) Music Enrichment (Cover Art) MusicID (File) Playlist, MoodGrid (10K Tracks)	6 MB	14 MB

© 2000 to present. Gracenote, Inc. All rights reserved.

Feature Set	Measured RAM Requirements*	Recommended RAM Requirements*
MusicID (CD, Text) Music Enrichment (Cover Art) MusicID (File) Playlist, MoodGrid (20K Tracks)	9 MB	20 MB
MusicID (CD, Text) Music Enrichment (Cover Art) MusicID (File) Playlist, MoodGrid (40K Tracks)	12 MB	32 MB

*MusicID (Text) lookups were done with a 5K track sample set in the feature set without Playlist and with 10K, 20K and 40K tracks for the corresponding Playlist size. MusicID (CD) lookup measurements were done using a 25K sample set of CD TOCs. MusicID (File) fingerprint lookups were done using 167 tracks from 11 albums.

Playlist Memory Usage

Memory usage for Playlist is directly correlated to the number of tracks available for playlisting. For smaller libraries, RAM requirements may be lower. For libraries with a larger number of songs, RAM requirements may increase. Memory usage increases at a rate of ~0.3 MB per thousand tracks.

Playlist relies on a Collection Summary of stored attribute data of a user's music collection. These collection summaries are designed to take up a minimal amount of disk space and are directly proportional to the user's collection size. Developers should take into account disk storage to persist these collection summaries which grow at the rate of ~110 KB per thousand tracks.

Performance

The performance numbers below are given only as reference data points and are not guaranteed, as many factors come into play in real systems, such as other competing processes, hardware limitations and the set of data being tested with. Performance measurements can have large variations depending on the platform used. The measurements below were done on a Beagle Board xM unless otherwise noted.

CD Lookup Performance

Local CD recognition through a CD TOC is relatively quick with local metadata available. Performance numbers for a local CD recognition can be well under a second. Online CD recognition is slower due to network latency and data transfer time. The following CD Lookup results were measured on the Beagle Board xM against the embedded North America (NA-0010) regional Gracenote Database (GDB) for local lookups. These measurements are represented as an average and max time for the 25K CD TOCs tested.

Lookup Type	Average Time (sec)	Max Time (sec)

© 2000 to present. Gracenote, Inc. All rights reserved.

Local CD Lookup	0.01	0.22
Local CD Lookup + Image	0.235	0.77
Online CD Lookup	0.3	12.4
Online CD Lookup + Image	0.8	30.24

Text Lookup Performance

Text recognition measurements were done using a sample set of 5000 MP3 tracks against both the NA-0010 and NA-0020 GDBs to demonstrate the difference in performance based on the size of the database. These measurements were taken on the Beagle Board xM and are represented as the total measured time for this 5000 track collection and does not include device enumeration (indexing).

	Total Recognition Time (sec)	
Lookup Type	**North America 120K Albums (NA-0010)**	**North America 30K Albums (NA-0020)**
Text Lookup	70	63
Text Lookup + Image	105	95

TrackID (Fingerprint) Performance

TrackID recognition measurements were done with the Return Single option from a sample set of 167 tracks and measured on the Beagle Board xM. Fingerprint generation was not included in the time below and takes ~1.4 seconds to complete before the online lookup can be done.

Lookup Type	**Average Time (s)**	**Max Time (s)**
TrackID Fingerprint Lookup	0.4	1.2
TrackID Fingerprint Lookup + Image	0.9	1.9

I/O Performance

The I/O numbers below were generated on the Beagle Board c4 platform and may be useful in determining I/O load on a device during lookup events. Both Local and Online scenarios include both Text and TOC lookups.

Lookup Type	**Average Number Reads/Query**	**Average Bytes Read/Query**	**Average Number Writes/Query**	**Average Bytes Written/Query**
Local Lookup	15	10 KB	N/A	N/A

© 2000 to present. Gracenote, Inc. All rights reserved.

Lookup Type	Average Number Reads/Query	Average Bytes Read/Query	Average Number Writes/Query	Average Bytes Written/Query
Local Lookup + Image	25	15 KB	N/A	N/A
Online Lookup	5	2 KB	30	15 KB
Online Lookup + Image	15	10 KB	100	45 KB

iOS

Deployment

GNSDK for Mobile 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 the Apple developer site mentioned above for further details.

Advanced Implementations

User History

The 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.

© 2000 to present. Gracenote, Inc. All rights reserved.

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 (IDNow)
- Recognizing music from a PCM sample (AlbumID)
- Searching by fingerprint

The process of adding an entry to the database is included in the search operation classes. This allows it to be called in the thread that invokes callbacks 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 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.

© 2000 to present. Gracenote, Inc. All rights reserved.

Mobile Getting Started

Getting Started with Mobile iOS

Introduction

The GNSDK for Mobile Software Development Kit (SDK) provides the GN_Music_SDK_iOS sample application that demonstrates Gracenote features.

The SDK also provides a development project as an example of how to incorporate the GNSDK for Mobile into your iPhone application. This document describes how to integrate the sample application project into your development environment.

For additional samples, contact your Gracenote Global Services & Support representative.

Fingerprint and Metadata Samples

Gracenote provides a local cache of fingerprints and metadata of some sample songs. The SDK uses this data to attempt a local ID prior to attempting an online lookup. The SDK provides song samples in the sample_music folder that you can use to test local lookup identification.

For additional samples, contact your Gracenote Global Services & Support representative.

Artist	Track	Album
Winterwood	Saturday	Love In The Heart
Winterwood	We Never Take The Time	Homeward Tonight

Setting Up the Development Environment

iOS development requires a specific environment. For details on setting up this environment, refer to the instructions at 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 GNSDK for Mobile sample application. To register, go to http://developer.apple.com/programs/register/

Finally, to distribute your application, you should get a provisioning certificate and file. Follow the instructions found at https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ProvisioningDevelopment.html

Requirements

The following are general requirements to build and run the sample application:

© 2000 to present. Gracenote, Inc. All rights reserved.

- Apple Developer Provisioning Certificate
- Apple App ID
- Apple Provisioning File
- iOS version 6.0 or higher
- Xcode version 5.0 or higher

Technical Requirements

Framework Size

The framework size reflects the combined size of the universal framework for all iOS architectures: armv7 and armv7s 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 your application does not utilize. 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 the armv7 and armv7s architectures. This includes the application code and all linked SDK code. If the application 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 strips unused symbols, reducing total application size.

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
- CoreData.framework (Required for the sample application only)
- CoreLocation (Required for the sample application only)
- AVFoundation.framework (must be weak-referenced in application)
- CoreMedia
- libstdc++

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

Linker Flags

The following linker flags **must** be used:

- -ObjC

Installing the iOS SDK

GNSDK for Mobile Sample Application uses the iOS v7 and XCode version 5 or later. Ensure your development environment supports these versions.

Follow the steps below to install the appropriate iOS SDK:

© 2000 to present. Gracenote, Inc. All rights reserved.

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 Requirements in the Release Notes). The download will also include the required version of Xcode.
3. Ensure Xcode and the 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.

Building and Running the Sample Applications

To run either of the sample applications:

1. Unpack the Gracenote iOS GNSDK for Mobile Package: GN_Music_SDK_<version>.zip
2. Launch Xcode.
3. Close the Welcome to Xcode launch screen.
4. Choose File > Open.
5. Navigate to GN_Music_SDK_iOS.
6. To run the sample application, add the ClientID and Client Tag strings provided by Gracenote to the project.
   a. For GN_Music_SDK_iOS, edit the file GN_Music_SDK_iOS-Prefix.pch file under Other Sources.

```
// For GN_Music_SDK_iOS, edit GN_Music_SDK_iOS-Prefix.pch

#define CLIENTID @"12345678"
 #define CLIENTIDTAG @"ABCDEFGHIJKLMNOPQRSTUVWXYZ012345"
```

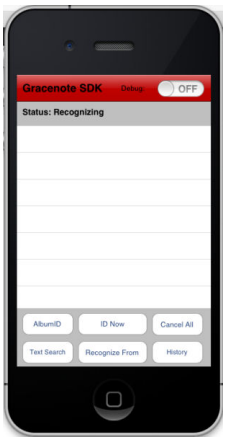
7. Add the Gracenote license file to the Xcode project.
8. Select the Overview drop down box at the top left corner of the application window.
9. In the first grouping, select Simulator.
10. Leave all other settings as is and click outside of the drop down box to exit it.
11. In the top middle of the window, click the Build and Run icon.
12. Look for Succeeded in the the bottom right of the window. If the build fails, see Troubleshooting.
13. The Simulator will open and the Gracenote iOS Sample Application will launch.

Testing the Sample Applications

To test the GN_Music_SDK_iOS sample application:

1. Tap on the buttons to perform the available queries. When testing the application, note that:
   - The Recognize From > FPX and Recognize From > PCM queries use inputs from the GN_ Music_SDK_iOS parent directory.
   - The Text Search and Lyric Fragment Search queries require you to manually input search terms.
2. Tap the ID Now button on the device to identify a song. You should see a screen similar to the following:

© 2000 to present. Gracenote, Inc. All rights reserved.



In the Sample Application, if debugging is enabled through the UI, logging will take place for everything except idNow operations. However, you will not see this issue if you enable debugging programmatically, See Using GNSDK for Mobile Debug Logging in the GNSDK for Mobile iOS Implementation Guide for more information.

© 2000 to present. Gracenote, Inc. All rights reserved.

Implementing Applications (All Platforms)

About the Implementing Applications Documentation

This section is intended as a general guide for applications using the object-oriented GNSDK, which supports development in a number of object-oriented languages such as C++, C#, Java, Android Java and Objective-C. This document discusses classes and methods in a generic sort of way while showing brief inline code snippets in different languages. For your particular language, some naming could be slightly different. Java, for example, uses camelcasing (first letter is lower-case). This is a small difference, but for other languages, like Objective-C, the differences could be more involved. To help with this, dropdown code samples in specific programming languages are provided.

For specific programming language implementation details, see the language's respective API reference documentation.

Basic Application Steps

To get started with GNSDK development, Gracenote recommends you follow these general steps, some of which are required and some of which are optional but recommended:

1. Required initialization—see *"Setup and Initialization"* :
   - **Get authentication**—Get a Client ID and license file from Gracenote Global Services & Support. These are used for initial authorization and in every query.
   - **Include necessary header files**—Include the header files and classes for your platform that your application requires.
   - **Initialize SDK** —Instantiate and initialize a GNSDK Manager object (`GnManager` class).
   - **Get a User Object** —Instantiate and initialize a User object (`GnUser`). All queries require a User object with correct Client ID information. You can create a new User, or deserialize an existing User, to get a User object.
2. Not required but suggested initialization:
   - **Enable logging**—Gracenote recommends that GNSDK logging be enabled to some extent. This aids in debugging applications and reporting issues to Gracenote. See *"Using Gracenote SDK Logging"* for more information.
   - **Load a locale(s)**—Gracenote recommends using locales as a convenient way to group locale-dependent metadata specific to a region (such as Europe) and language that should be returned from the Gracenote service. In addition, some metadata requires having a locale loaded. See *"Loading a Locale"* for more information.
   - **Enable GNSDK storage and caching**—See "Enabling and Using GNSDK Storage" for more information.

© 2000 to present. Gracenote, Inc. All rights reserved.

3. If you are using a local database of fingerprints and metadata for identification, then download it and save it to local storage. Set the appropriate mode for online and local lookups. See "Using a Local Fingerprint Database" and "Setting Local and Online Lookup Modes" for more information.
4. Perform queries to identify music and video. See "Identifying Music" and *Identifying Video* for more information.
5. Parse query result objects for desired metadata. See "Processing Returned Metadata Results" for more information.
6. Exit the application when done, no cleanup necessary.

Setup and Initialization

To get started with GNSDK development:

1. Get a Client ID and License file for application authentication from Gracenote Global Services & Support. See Authorizing a GNSDK Application.
2. Include the GNSDK header for your platform to include all necessary libraries and headers. See Including Header Files.
3. Instantiate a `GnManager` object. See Instantiating a GNSDK Manager Object.
4. Instantiate a `GnUser` user object. See Instantiating a User Object.

Authorizing a GNSDK Application

Gracenote uses product licensing and server-side entitlements to manage your application's access to metadata.

As a Gracenote customer, Gracenote Global Services & Support works with you to determine the kind of products you need (such as MusicID, Playlist, and so on). Gracenote also determines the metadata your application requires for the products you license.

Gracenote uses this information to create a unique customer ID (called a Client ID), a license file, and server-side metadata entitlements specifically tailored to your application.

When developing a GNSDK application, you must include a Client ID and license file to authorize your application with Gracenote. In general, the License file enables your application to use the Gracenote products (and their corresponding GNSDK modules) that you purchased. Gracenote Media Services uses the Client ID to enable access to the metadata your application is entitled to use.

All applications are entitled to a set of core metadata based on the products that are licensed. Your application can access enriched metadata through server-side metadata entitlements. Contact your Gracenote Global Services & Support representative for more information.

Some applications require a local (embedded) database for metadata. These systems do not access Gracenote Media Services to validate metadata entitlements and access metadata. Instead, metadata entitlements are pre-applied to the local database.

© 2000 to present. Gracenote, Inc. All rights reserved.

Client ID

Each GNSDK customer receives a unique Client ID string from Gracenote. This string uniquely identifies each application to Gracenote Media Services and lets Gracenote deliver the specific metadata the application requires.

A Client ID string consists of two sets of numbers separated with a hyphen, e.g., 123456-789123456789012312.

License File

Gracenote provides a license file along with your Client ID. The license file notifies Gracenote to enable the GNSDK products you purchased for your application.

You should secure your Gracenote client id, tag and license information. Something similar to the way Android recommends protecting a Google Play public key from malicious hackers: http://developer.android.com/google/play/billing/billing_best_practices.html

Including Header Files

GNSDK consists of a set of shared modules. The GNSDK Manager module is required for all applications. All other modules are optional. Your application's feature requirements determine which additional modules should be used.

For convenience, all your application has to do is include a single GNSDK header file and all necessary header files and libraries will be automatically included.

Java/Android Java

```
import com.gracenote.gnsdk.*;
```

Objective-C

```
#import <GnSDKObjC/Gn.h>
```

Windows Phone C#

```
using Gracenote;
```

Instantiating a GNSDK Manager Object

The first thing your application needs to do is initialize an SDK Manager object (`GnManager`) using the GNSDK library path and the contents of the license file you obtained from GGSS. The SDK Manager object is used to monitor an application's interaction with Gracenote.

Specifying the License File

Your application must provide the license file when you allocate a `GnManager` object. This class' constructor gives you the following options for submitting the license file:

© 2000 to present. Gracenote, Inc. All rights reserved.

- **Null-terminated string**—Set the input mode parameter to `GnLicenseInputMode.kLicenseInputModeString` and pass the license file as a null-terminated string (see examples below).
- **Filename**—Set the input mode parameter to `GnLicenseInputMode.kLicenseInputModeFilename` and pass the relative filename in the string parameter.

Code samples

Android Java

```
private String gnsdkLicense = <get license as string from asset>;
Context context = this.getApplicationContext();

// Initialize GNSDK
gnsdk = new GnManager(context,                                        // Android Context
                      gnsdkLicense,                                   // License as a string
                      GnLicenseInputMode.kLicenseInputModeString);  // Input License as a string
```

Objective-C

```
@property  (strong) GnManager *gnManager;
NSError*   error = nil;
NSString*  resourcePath  = [[NSBundle mainBundle] pathForResource:
                            gnsdkLicenseFilename ofType: nil];
NSString*   licenseString = [NSString stringWithContentsOfFile: resourcePath
                             encoding: NSUTF8StringEncoding
                             error: &error];
self.gnManager = [[GnManager alloc] initWithLicense: licenseString licenseInputMode:
kLicenseInputModeString];
```

Windows Phone C#

```
/*
* Initalize GNSDK
*/
string licenseString = ReadFile(App.gnLicenseFileName_);
App.gnManager_ = new GnManager(licenseString, GnLicenseInputMode.kLicenseInputModeString);
```

Instantiating a User Object

To make queries, every application is required to instantiate a User object (`GnUser`). Most devices will only have one user; however, on a server, for example, there could be a number of users running your application. Gracenote uses the Client ID and Client Tag to verify that the licensed and allowable users quota has not been exceeded.

If an application cannot go online at user-registration, but has a local fingerprint database it can use for identification, it has the option to create and use a 'local only' User object. A User object created with online connectivity can do both local and online queries. To determine if the User object can only do local queries, your application can call the `GnUser`'s `IsLocalOnly` method.

`GnUser` objects can be created 'online,' which means the Gracenote back-end creates and verifies them. Alternatively, they can be created 'local only,' which means the SDK creates and uses them locally.

For example (Java):

© 2000 to present. Gracenote, Inc. All rights reserved.

```
// Create user for video and music
gnUser = new GnUser(gnUserStore, clientId, clientTag, "1.0");
```

Saving the User Object to Persistent Storage

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

To save to persistent storage, you have the option to implement the `IGnUserStore` interface which requires you to implement two methods: `LoadSerializedUser` and `StoreSerializedUser`.

On mobile and ACR platforms (Android, iOS, Windows), the SDK provides the `GnUserStore` class, a platform-specific implementation of the `IGnUserStore` interface. Storage on these devices is implemented in platform-specific ways. On Android, for example, the User object is saved to shared preferences.

You can then pass an instance of this class as a parameter in a `GnUser` constructor and the SDK will automatically read the User object from storage and use it to create a new User ID. The SDK may also periodcally `SaveSerializedUser` when user data changes.

Allocating a User object code samples:

Android Java code sample

```
// Provide Android Context to GnUserStore
GnUser gnUser = new GnUser( new GnUserStore(context), clientId, clientTag, appString );
```

Objective-C

```
@property (strong) GnUser *gnUser;
self.gnUserStore = [[GnUserStore alloc] init];
self.gnUser = [[GnUser alloc] initWithGnUserStoreDelegate: self.gnUserStore
                clientId: clientID
                clientTag: clientIDTag
                applicationVersion: @"1.0.0.0"];
```

Windows Phone C#

```
// Create ACR User
if (null == App.gnUser_)
{
   App.gnUser_ = new GnUser(new GnUserStore(App.gnClientId_),
                 App.gnClientId_,
                 App.gnClientIdTag_,
                 App.applicationVersion_);
   App.gnUser_.Options.LookupMode(GnLookupMode.kLookupModeOnline);
}
```

© 2000 to present. Gracenote, Inc. All rights reserved.

Loading a Locale

GNSDK provides *locales* as a convenient way to group locale-dependent metadata specific to a region (such as Europe) and language that should be returned from the Gracenote Service. A locale is defined by a group (such as Music), a language, a region and a descriptor (indicating level of metadata detail), which are identifiers to a specific set of lists in the Gracenote Service.

Using locales is relatively straightforward for most applications to implement. However, it is not as flexible or complicated as accessing lists directly - most locale processing is handled in the background and is not configurable. For most applications though, using locales is more than sufficient. Your application should only access lists directly if it has a specific reason or use case for doing so.

To load a locale, allocate a `GnLocale` object with one of the class constructors. The `GnLocale` constructors take parameters indicating the following:

- **Group** - Group type of locale such as Music, Playlist or Video that can be easily tied to the application's use case
- **Region** - Region the application is operating in, such as US, China, Japan, Europe, and so on, possibly specified by the user configuration
- **Language** - Language the application uses, possibly specified by the user configuration
- **Descriptor** - Additional description of the locale, such as Simplified or Detailed for the list hierarchy group to use, usually determined by the application's use case
- **Status callback (optional)** - One of the constructors takes a `GnStatusEventsListener` callback object

Locales have the following space requirements: 2MB for a music only locale, 6MB for a music and playlist locale.

For example:

- A locale defined for the USA of English/ US/Detailed returns detailed content from a list written in English for a North American audience.
- A locale defined for Spain of Spanish/Global/Simplified returns list metadata of a less-detailed nature, written in Spanish for a global Spanish-speaking audience (European, Central American, and South American).

Windows Phone C#

```
GnLocale locale =
         new GnLocale(GnLocaleGroup.kLocaleGroupMusic,
                      GnLanguage.kLanguageEnglish,
                      GnRegion.kRegionGlobal,
                      GnDescriptor.kDescriptorDefault,
                      App.mGnUser,
                      null
                  );

locale.SetGroupDefault();
```

© 2000 to present. Gracenote, Inc. All rights reserved.

Locale-Dependent Data

Thr following metdata fields require having a locale loaded:

- artist type - levels 1-2
- era - levels 1-3
- genre - levels 1-3
- mood - levels 1-2
- origin - levels 1-4
- tempo - levels 1-3
- role
- role category
- entity type
- composition form
- instrumentation
- package display language

Default Regions and Descriptors

When loading a locale, your application provides inputs specifying group, language, region, and descriptor. Region and descriptor can be set to “default.”

When no locales are present in the local database, or no local database is enabled, and the application is configured for online access, GNSDK uses the Global region when the default region is specified, and the Detailed descriptor when the default descriptor is specified.

Otherwise, when “default” is specified, GNSDK filters the local database and loads a locale matching the group and language (and the region and descriptor, if they are not specified as default). Complete locales (those with all sub-components present) are preferred over incomplete locales. If, after filtering, the local database contains multiple equally complete locales, a default locale is chosen using the defaults shown in the table below:

Regional GDB	Available Locales	Default Locale
North America (NA)	US and Latin America	US
Latin America (LA)	Latin America	Latin America
Korea (KR)	Korea	Korea
Japan (JP)	Japan	Japan
Global (GL)	Global, Japan, US, China, Taiwan, Korea, Europe, and Latin America	Global
Europe (EU)	Europe	Europe
China (CN)	China and Taiwan	China

If no locales are present after filtering the local database, an error is returned.

© 2000 to present. Gracenote, Inc. All rights reserved.

Default regions and descriptors can be used to write generic code for loading a locale. For example, consider an application targeted for multiple devices: one with a small screen, where the Simplified locales are desired; and one with a large screen, where more detail can be displayed to the user, and the Detailed locales are desired. The application code can be written to generically load locales with the “default” descriptor, and each application can be deployed with a local database containing simplified locales (small-screen version), or detailed locales (large-screen version). GNSDK loads the appropriate locales based on the contents of the local database.

Locale Groups

Setting the locale for a group causes the given locale to apply to a particular media group - Music, Playlist, Video or EPG. For example, setting a locale for the Music group applies the locale to all music-related objects. When a locale is loaded, all lists necessary for the locale group are loaded into memory. For example, setting the locale for the Playlist group causes all lists needed to generate playlists to be loaded.

Once a locale has been loaded, you must call the `GnLocale`'s `SetGroupDefault` method before retrieving locale-dependent values.

Multi-Threaded Access

Since locales and lists can be accessed concurrently, your application has the option to perform such actions as generating a Playlist or obtaining result display strings using multiple threads.

Typically, an application loads all required locales at start up, or when the user changes preferred region or language. To speed up loading multiple locales, your application can load each locale in its own thread.

Updating Locales and Lists

GNSDK supports storing locales and their associated lists locally, which improves access times and performance. Your application must enable a database module (such as SQLite) to implement local storage. For more information, See "Enabling and Using GNSDK Storage" .

Update Notification

Periodically, your application may need to update any locale lists that are stored locally. As a best practice, Gracenote recommends registering a locale update notification callback or, if you are using lists directly, a lists update nortification callback. To do this, you need to code an `IGnSystemEvents` delegate that implements the locale or list update methods—`LocaleUpdateNeeded` or `ListUpdateNeeded`—and provide that delegate as a parameter to the `GnManager`'s `EventHandler` method. When GNSDK internally detects that a locale or list is out of date, it will call the appropriate callback. Detection occurs when a requested list value is not found. This is done automatically without the need for user application input. Note, however, that if your application does not request locale-dependent metadata or missing locale-dependent data, no detection will occur.

Updates require the user lookup mode option to be set to online lookup - `kLookupModeOnline`(default) or online lookup only—`kLookupModeOnlineCacheOnly`. This allows the SDK to retreive locales from the Gracenote Service. You may need to toggle this option value for the update process. For more information about setting the user option, "Setting Local and Online Lookup Modes".

© 2000 to present. Gracenote, Inc. All rights reserved.

Once your app receives a notification, it can choose to immediately do the update or do it later. Gracenote recommends doing it immediately as this allows the current locale/list value request to be fulfilled, though there is a delay for the length of time it takes to complete the update process.

The GNSDK does not set an internal state or persistant flag indicating an update is required; your application is responsibile for managing the deferring of updates beyond the notification callback.

Locale Behavior

How locales are stored, accessed and updated depends on how you have configured your storage and lookup options as shown in the following table. For information on configuring lookup modes see "Setting Local and Online Lookup Modes".

Storage Provider Initialized	GnLookupMode Enum	Behavior
Either	`kLookupModeOnlineNoCache`	Locales are always downloaded and stored in RAM, not local storage.
Not initialized	`kLookupModeOnline`	Locales are always downloaded and stored in RAM, not local storage.
Initialized	`kLookupModeOnline`	If downloaded, locales are read from local storage. Downloaded locales are written immediately to local storage.
Not Initialized	`kLookupModeOnlineNoCacheRead`	Locales are always downloaded and stored in RAM.
Initialized	`kLookupModeOnlineNoCacheRead`	Locales are always downloaded and stored in RAM and local storage. Locale data is always read from RAM, not local storage.
Initialized	`kLookupModeOnlineCacheOnly` `kLookupModeLocal`	Locale data is read from local storage. If requested data is not in locale storage the load attempt fails. Local storage is updated when new versions become available. The application developer is responsible for providing that mechanism.

Locale behavior may not change if the lookup mode is changed after the locale is loaded. For example, if a locale is loaded when the lookup mode is `kLookupModeOnline`, locale data will be read from local storage even if the lookup mode is changed.

© 2000 to present. Gracenote, Inc. All rights reserved.

Best Practices

Practice	Description
Applications should use locales.	Locales are simpler and more convenient than accessing lists directly. An application should only use lists if there are specific circumstances or use cases that require it.
Apps should register a locale/list update notification callback and, when invoked, immediately update locales/lists.	See the ***Update Notification*** section above.
Applications can deploy with pre-populated list stores and reduce startup time.	On startup, a typical application loads locale(s). If the requested locale is not cached, the required lists are downloaded from the Gracenote Service and written to local storage. This procedure can take time. Customers should consider creating their own list stores that are deployed with the application to decrease the initial startup time and perform a locale update in a background thread once the application is up and running.
Use multiple threads when loading or updating multiple locales.	Loading locales in multiple threads allows lists to be fetched concurrently, reducing overall load time.
Update locales in a background thread.	Locales can be updated while the application performs normal processing. The SDK automatically switches to using new lists as they are updated. ⚠ If the application is using the GNSDK Manager Lists interface directly and the application holds a list handle, that list is not released from memory and the SDK will continue to use it.
Set a persistence flag when updating. If interrupted, repeat update.	If the online update procedure is interrupted (such as network connection/power loss) then it must be repeated to prevent mismatches between locale required lists. Your application should set a *persistence* flag before starting an update procedure. If the flag is still set upon startup, the application should initiate an update. You should clear the flag after the update has completed.

© 2000 to present. Gracenote, Inc. All rights reserved.

Practice	Description
Call the `GnManager::StorageCompact` method after updating lists or locales.	As records are added and deleted from locale storage, some storage solutions, such as SQLite, can leave empty space in the storage files, artificially bloating them. You can call the `GnManager::StorageCompact` method to remove these. The update procedure is not guaranteed to remove an old version of a list from storage immediately because there could still be list element references which must be honored until they are released. Therefore, your application should call the `GnSDK::StorageCompact` method during startup or shutdown after an update has finished.

Local only applications should set the user handle option for lookup mode to local only.	If your application wishes to only use the Locales in pre-populated Locales storage, then it must set the user handle lookup mode to local.
	
	For example (C++)
	
	`/* Set lookup mode (all queries done with this user will inherit the lookup mode) */`
	`user.Options().LookupMode(kLookupModeLocal);`

To simplify the implementation of multi-region applications, use the default region and descriptor.	The Locale subsystem can infer a region and descriptor from the Locale store that can be used in place of the region and descriptor deaults when loading a locale. This can simplify implementing an application intended to be deployed in different regions with its own region specific pre-populated Locale store. If you are deploying your app to multiple regions with a pre-populated Locale store containing locales for all target regions then you should use `kRegionDefault` and `kDescriptorDefault` when loading a locale. In this case, the same region and descriptor are used based on defaults hardcoded into the SDK.

Using Gracenote SDK Logging

The `GnLog` class has methods to enable Gracenote SDK logging, set options, write to the SDK log, and disable SDK logging.

There are 3 approaches you can take to implementing logging using the GNSDK:

1. **Enable GNSDK logging**—This creates log file(s) that you and Gracenote Global Services & Support can use to evaluate and debug any problems your application might encounter when using the SDK
2. **Enable GNSDK logging and add to it**—Use the `GnLog Write` method to add your application's

© 2000 to present. Gracenote, Inc. All rights reserved.

log entries to the GNSDK logs.

3. **Implement your own logging mechanism (via the logging callback)**— Your logging callback, for example, could write to the console, Unix Syslog, or the Windows Event Log.

> Gracenote recommends you implement callback logging (see Implementing Callback Logging). On some platforms, e.g., Android, GNSDK logging can cause problematic system delays. Talk to your GGSS representative for more information.

Enabling GNSDK Logging

To use Gracenote SDK logging:

1. Instantiate a `GnLog` object.

   This class has two constructors: both require you to set a log file name and path, and a `IGnLogEvents` logging callback delegate (`GnLogEventsDelegate` in Objective-C). One of the constructors also allows you to set logging options, which you can also set via class methods. These include ones for:

   - What type of messages to include: error, warning, information or debug (`GnLogFilters`)
   - What fields to log: timestamps, thread IDs, packages, etc. (`GnLogColumns`)
   - Maximum size of the log file in bytes, synchronous or asynchronous logging, and archive options (`GnLogOptions`)

2. Call the `GnLog Enable(PackageID)` method to enable logging for specific packages or all packages.

   > A *package* is a GNSDK Library as opposed to a module, which is a block of functionality within a package. See the `GnLogPackageType` enums for more information on GNSDK packages.

   > Note that `Enable` returns its own `GnLog` object to allow method chaining.

3. Call the `GnLog Write` method to write to the GNSDK log
4. Call the `GnLog Disable(PackageID)` method to disable logging for a specific package or all packages.

Logging code samples

Java and Android Java

```
// Enable GNSDK logging
String gracenoteLogFilename = Environment.getExternalStorageDirectory().getAbsolutePath() +
File.separator + gnsdkLogFilename;
gnLog = new GnLog(gracenoteLogFilename, null);
gnLog.columns(new GnLogColumns().all());
gnLog.filters(new GnLogFilters().all());
gnLog.enable(GnLogPackageType.kLogPackageAll);
```

© 2000 to present. Gracenote, Inc. All rights reserved.

Objective-C

```
NSString *docsDir = [GnAppDelegate applicationDocumentsDirectory];
docsDir = [docsDir stringByAppendingPathComponent:@"log.txt"];

self.gnLog = [[GnLog alloc] initWithLogFilePath:docsDir
filters:[[[GnLogFilters alloc]init]all]
columns:[[[GnLogColumns alloc]init]all]
options:[[[GnLogOptions alloc]init]maxSize:0]
logEventsDelegate:self];

// Max size of log: 0 means a new log file will be created each run
[self.gnLog options: [[[GnLogOptions alloc] init]maxSize:0]];
[self.gnLog enableWithPackage:kLogPackageAllGNSDK error:nil];
```

C# and Windows Phone C#

```
/* Enable GNSDK logging */
App.gnLog_ = new GnLog(Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path,
"sample.log"), (IGnLogEvents)null);

App.gnLog_.Columns(new GnLogColumns().All);
App.gnLog_.Filters(new GnLogFilters().All);

GnLogOptions options = new GnLogOptions();
options = options.MaxSize(0);
options = options.Archive(false);
App.gnLog_.Options(options);

App.gnLog_.Enable(GnLogPackageType.kLogPackageAll);
```

The GNSDK logging system can manage multiple logs simultaneously. Each call to the enable API can enable a new log, if the provided log file name is unique. Additionally, each log can have its own filters and options.

Implementing Callback Logging

You also have the option to direct GNSDK to allow a logging callback, where you can determine how best to capture and disseminate specific logged messages. For example, your callback function could write to its own log files or pass the messages to an external logging framework, such as the console, Unix Syslog, or the Windows Event Log.

Enabling callback is done with the `GnLog` constructor where you have the option to pass it a `IGnLogEvents` (`GnLogEventsDelegate` in Objective-C) callback, which takes callback data, a package ID, a filter mask, an error code, and a message field.

Enabling and Using GNSDK Storage

To improve performance, your application can enable internal GNSDK storage and caching. The GNSDK has two kinds of storage, each managed through a different class:

1. **Online stores for lookups**—The GNSDK generates these as lookups take place. Use `GnStoreOps` methods to manage these.
2. **Local lookup databases**—Gracenote Global Support & Services (GGSS) generates these databases, which differ based on region, configuration, and other factors, and ships them to

© 2000 to present. Gracenote, Inc. All rights reserved.

customers as read-only files. These support TUI, TOC and text lookup for music searches. The `GnLookupLocal` class can be used to manage these databases.

To enable and manage GNSDK storage:

1. Enable a *storage provider* (SQLite) for GNSDK storage
2. Allocate a `GnManager` object for online stores (optional)
3. Allocate a `GnLookupLocal` object for local lookup databases (optional)
4. Set a folder location(s) for GNSDK storage (required)
5. Manage storage through `GnManager` and `GnLookupLocal` methods

Enabling a Provider for GNSDK Storage

Before GNSDK storage can take place, you need to enable a storage provider. Right now, that means using the GNSDK SQLite module. Note that this is for GNSDK use only—your application cannot use this database for its own storage.

For information on using SQLite, see http://sqlite.org.

In the future, other database modules will be made available, but currently, the only option is SQLite.

To enable local storage, you need to call the `GnStorageSqlite`'s `Enable` method which returns a `GnStorageSqlite` object.

C++ code snippet:

```
/* Enable StorageSQLite module to use as our database engine */
GnStorageSqlite& storageSqlite = GnStorageSqlite::Enable();
```

Java/Android Java

```
GnStorageSqlite gnStorageSqlite = GnStorageSqlite.enable();
```

Objective-C

```
self.gnStorageSqlite = [GnStorageSqlite enable: &error];
```

Windows Phone C#

```
App.gnStorageSqlite_ = GnStorageSqlite.Enable;
```

GNSDK Stores

Once enabled, the GNSDK manages these stores:

Stores	Description
Query store	The query store caches media identification requests
Lists store	The list store caches Gracenote display lists

© 2000 to present. Gracenote, Inc. All rights reserved.

Stores	Description
Content store	The content stores caches cover art and related information

You can get an object to manage these stores with the following `GnManager` methods:

- `GnStoreOps& QueryCacheStore`—Get an object for managing the query cache store.
- `GnStoreOps& LocalesStore`—Get an object for managing the locales/lists store.
- `GnStoreOps& ContentStore`—Get an object for managing the content store.

GNSDK Databases

Once enabled, the GNSDK manages these databases as the following `GnLookupLocal::GnLocalStorageName` enums indicate:

Database (GnLookupLocal)	Description
`kLocalStorageContent`	Used for querying Gracenote content.
`kLocalStorageMetadata`	Used for querying Gracenote metadata.
`kLocalStorageTOCIndex`	Used for CD TOC searching.
`kLocalStorageTextIndex`	Used for text searching.

Setting GNSDK Storage Folder Locations

You have the option to set a folder location for all GNSDK storage or locations for specific stores and databases. You might, for example, want to set your stores to different locations to improve performance and/or tailor your application to specific hardware. For example you might want your locale list store in flash memory and your image store on disk.

If no locations for storage are set, the GNSDK, by default, uses the current directory. To set a location for all GNSDK storage, use the `GnStorageSqlite StorageLocation` method.

Use the `GnStoreOps`' `Location` method or the `GnLookupLocal`'s `StorageLocation` method to set specific store or database locations. `StorageLocation` takes a database enum and a path location string. To set a store location, you would need to allocate a `GnStoreOps` object for a specific cache using `GnManager` methods and call its `Location` method.

Java/Android Java

```
gnStorageSqlite.storageLocation(getExternalFilesDir(null).getAbsolutePath());
```

Objective-C

```
[self.gnStorageSqlite storageLocationWithFolderPath:[GnAppDelegate applicationDocumentsDirectory]
error: &error];
```

Windows Phone C#

```
App.gnStorageSqlite_.StorageLocation(Windows.Storage.ApplicationData.Current.LocalFolder.Path);
```

© 2000 to present. Gracenote, Inc. All rights reserved.

For Windows CE, an absolute path must be used in setting storage folder location

Getting Local Lookup Database Information

You can retrieve manifest information about your local databases, including database versions, available image sizes, and available locale configurations. Your application can use this information to request data more efficiently. For example, to avoid making queries for unsupported locales, you can retrieve the valid locale configurations contained in your local lists cache.

Image Information

GNSDK provides album cover art, and artist and genre images in different sizes. You can use the `kImageSize` key with the `GnLookupLocal`'s `StorageInfo` method to retrieve available image sizes. This allows you to request images in available sizes only, rather than spending cycles requesting image sizes that are not available.

Use the `GnLookupLocal`'s `StorageInfoCount` method to provide ordinals to the `StorageInfo` method to get the image sizes.

Database Versions

To retrieve the version number for a local database, use the `kGDBVersion` key with the `StorageInfo` method. Use an ordinal of 1 to get the database version.

C++ example:

```
gnsdk_cstr_t  gdb_version = gnLookupLocal.StorageInfo(kMetadata, kGDBVersion, ordinal);
```

Getting Available Locales

Use the `GnLocale`'s `LocalesAvailable` method to get valid locale configurations available in your local lists store. Locale configurations are combinations of values that you can use to set the locale for your application. This method returns values for group, region, language and descriptor. Returns a count of the values available for a particular local database and local storage key.

Setting Online Cache Expiration

You can use the `GnUser`'s `CacheExpiration` method to set the maximum duration for which an item in the GNSDK query cache is valid. The duration is set in seconds and must exceed one day (> 86400). Setting this option to a zero value (0) causes the cache to start deleting records upon cache hit, and not write new or updated records to the cache; in short, the cache effectively flushes itself. The cache will start caching records again once this option is set to a value greater than 0. Setting this option to a value less than 0 (for example: -1) causes the cache to use default expiration values.

Managing Online Cache Size and Memory

You can use the following `GnStorageSqlite` methods to manage online cache size on disk and in memory:

© 2000 to present. Gracenote, Inc. All rights reserved.

- **`MaximumSizeForCacheFileSet`**—Sets the maximum size the GNSDK cache can grow to in kilobytes; for example "100" for 100 Kb or "1024" for 1 MB. This limit applies to each cache that is created. If the cache files' current size already exceeds the maximum when this option is set, then the passed maximum is not applied. When the maximum size is reached, new cache entries are not written to the database. Additionally, a maintenance thread is run that attempts to clean up expired records from the database and create space for new records. If this option is not set the cache files default to having no maximum size.
- **`MaximumMemorySizeForCacheSet`**— Sets the maximum amount of memory SQLite can use to buffer cache data. The value passed for this option is the maximum number of Kilobytes of memory that can be used. For example, "100" sets the maximum to 100 KB, and "1024" sets the maximum to 1 MB.

Setting Local and Online Lookup Modes

You can set lookup modes to determine if GNSDK lookups are done locally or online. GNSDK is designed to operate exactly the same way in either case. You can use the `GnUser.LookupMode` method to set this option for the user. You can also set this option for specific queries.

The terms *local* and *online* apply to the following:

1. **Online lookup**—Refers to queries made to the Gracenote service over the Internet.
2. **Online queries stored locally**—The GNSDK generates these as lookups take place. Even though they are stored locally, online stores are considered part of online lookup, not local lookup. The `GnManager` class can be used to manage these stores. Note that this store requires your application to enable GNSDK storage and caching. See *"Enabling and Using GNSDK Storage"* for more information.
3. **Local lookup fingerprint and metadata database**—Gracenote provides a mechanism for using a local database of fingerprints and metadata for identification. The `GnLookupFpLocal` class can be used for this database. See *Using a Local Fingerprint Database* for more information on this topic.

Supported Lookup Modes

GNSDK supports the following lookup mode options as shown with these `GnLookupMode` enums:

- **`kLookupModeOnline`**—This is the default lookup mode. If a cache exists, the query checks it first for a match. If no match is found in the cache, then an online query to the Gracenote Service is performed. If a result is found there, it is stored in the local online cache. If no connection to the Gracenote Service exists, the query will fail. The length of time before cache lookup query expires can be set via the user object.
- **`kLookupModeLocal`**—This mode forces the lookup to be done against any local databases only. Local stores created from (online) query results are not queried in this mode. If no local database exists, the query will fail.

The local and online modes are the standard modes for applications to choose between. The other online options (`kLookupModeOnlineNoCache`, `kLookupModeOnlineNoCacheRead`, and `kLookupModeOnlineCacheOnly`) are variations of the online mode. These additional online lookup modes give more control over when the SDK is allowed to perform a network connection and how to use the

© 2000 to present. Gracenote, Inc. All rights reserved.

online queries stored locally. The online-query store is used as a performance aid for online queries. If no storage provider is present, no online-query store is utilized.

Setting lookup mode for user code sample (C++):

```
/* Set lookup mode (all queries done with this user will inherit the lookup mode) */
user.Options().LookupMode(kLookupModeLocal);
```

Objective-C:

```
NSError *error = nil;
[[self.gnUser options] lookupModeWithLookupMode:kLookupModeLocal error:&error];
```

Default Lookup Mode

If the application does not set one, the GNSDK sets a default lookup mode—`kLookupModeOnline`—unless the GNSDK license file limits all queries to be local-only, which prevents the SDK from connecting online. When this limit is set in the license file, the lookup mode defaults to `kLookupModeLocal`.

Setting the Lookup Mode for a User or Specific Query

You can set the lookup mode as a user option or set it separately as a specific query option. Calling the `GnUser.LookupMode` method applies the option to all queries using the user handle. You can also use the `GnMusicId.LookupMode`, `GnMusicIdFile.LookupMode`, or `GnMusicIdStream.LookupMode` methods to override this for specific queries.

User example (C++):

```
GnUser user = GnUser(userStore, clientId, clientIdTag, applicationVersion);
/* Set user to match our desired lookup mode (all queries done with this user will inherit the
lookup mode) */
user.Options().LookupMode(kLookupModeLocal);
```

Query example (C++):

```
/* Perform the query */
music_id.Options().LookupMode(kLookupModeLocal);
GnResponseAlbums response = music_id.FindAlbums(albObject);
```

Query example (Objective-C):

```
GnMusicIdStreamOptions *options = [self.gnMusicIDStream options];
[options lookupMode:kLookupModeLocal error:&error];
```

Using Both Local and Online Lookup Modes

Your application can switch between local and online lookups, as needed.

Using a Local Fingerprint Database

Gracenote provides a mechanism for you to use a local database of track fingerprints and metadata for identification. The SDK can use this database to attempt a local ID prior to going online. Doing this provides a significant performance improvement if a local match is found.

© 2000 to present. Gracenote, Inc. All rights reserved.

To create a local database, you need to download and *ingest* raw fingerprint and metadata from Gracenote, packaged in an entity called a *bundle*. Currently, you have to work with Gracenote Global Support & Services (GGSS) to create a bundle. In the future, Gracenote will provide a self-service tool to do this. Having a local database requires that a storage provider, such as SQLite be enabled. You can do this with the `GnStorageSqlite` class.

Your application can ingest multiple bundles. If the same track exists in multiple ingested bundles it is added to the local database only once with the most recent/up-to-date track information. The ingestion process can be lengthy; your application may want to do this on a background thread to avoid stalling the main application thread.

To implement local lookup, use the following `GnLookupLocalStream` methods :

- **`Enable`** —Enables the `LookupLocalStream` library.
- **`StorageLocation`** —Sets the location for your MusicID-Stream fingerprint database . Bundle ingestion creates the database at the location specified. If the path does not exist, ingestion will fail.
- **`StorageRemove`** —Removes an item from the local fingerprint database.
- **`StorageClear`** —Clears all tracks from the local fingerprint database.

And the following `GnLookupLocalStreamIngest` methods:

- **`Write`** —Takes byte data and writes specified number of bytes to local storage.
- **`Flush`** —Flushes the memory cache to the file storage and commits the changes. This method ensures that everything written is commited to the file system. Note that this is an optional call as internally data is flushed when it exceed the cache size and when the object goes out of scope.

Note that the `GnLookupLocalStreamIngest` constructor takes a callback object to handle statuses as the ingest process can take some time. See the API reference documentation for `GnLookupLocalStreamIngest` for more information.

Downloading and Ingesting Bundles

To manage bundles, your application would typically need to do the following:

- Manually retrieve a Gracenote-provided bundle
- Place bundle in an online location that your application can access

  Have your application download and ingest the bundle.
-

> Note: Bundles should be retrieved from an online source. Gracenote recommends that when your application is installed or initialized that it download and ingest the latest bundle rather than ship with a bundle as part of the application binaries.

Ingesting bundles code samples:

Android Java

© 2000 to present. Gracenote, Inc. All rights reserved.

```
// Enable storage provider allowing GNSDK to use its persistent stores
GnStorageSqlite.enable();

// Enable local MusicID-Stream recognition (GNSDK storage provider must be enabled as pre-requisite)
GnLookupLocalStream.enable();

// Ingest MusicID-Stream local bundle, perform in another thread as it can be lengthy
Thread ingestThread = new Thread( new LocalBundleIngestRunnable(context) );
ingestThread.start();

/**
 * Loads a local bundle for MusicID-Stream lookups
 */
class LocalBundleIngestRunnable implements Runnable {
   Context context;

   LocalBundleIngestRunnable(Context context) {
          this.context = context;
   }

   public void run() {
       try {

           // Our bundle is delivered as a package asset
           // to ingest the bundle access it as a stream and write the bytes to
           // the bundle ingester.
           // Bundles should not be delivered with the package as this, rather they
           // should be downloaded from your own online service.

           InputStream  bundleInputStream = null;
           int          ingestBufferSize  = 1024;
           byte[]       ingestBuffer      = new byte[ingestBufferSize];
           int          bytesRead         = 0;

           GnLookupLocalStreamIngest ingester = new GnLookupLocalStreamIngest(new BundleIngestEvents
());

           try {

               bundleInputStream = context.getAssets().open("1557.b");

               do {

                   bytesRead = bundleInputStream.read(ingestBuffer, 0, ingestBufferSize);
                   if ( bytesRead == -1 )
                      bytesRead = 0;

                   ingester.write( ingestBuffer, bytesRead );

                } while( bytesRead != 0 );

     } catch (IOException e) {
          e.printStackTrace();
     }

     ingester.flush();

   } catch (GnException e) {
       Log.e( appString, e.errorCode() + ", " + e.errorDescription() + ", " + e.errorModule() );
   }
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
  }
}
```

Objective-C

```
- (NSError *) setupLocalLookup
{
    NSError *    error = nil;

    /*    Initialize the local lookup so we can do local lookup queries.    */
    self.gnLookupLocalStream = [GnLookupLocalStream enable: &error];
    if (! error)
    {
        NSString *    docDir = [GnAppDelegate applicationDocumentsDirectory];
        [self.gnLookupLocalStream storageLocation: docDir
                                            error: &error];
        if (! error)
        {
                // Look for the 10,000 track bundle and if not found try the little one.
                NSString*    bundlePath = [[NSBundle mainBundle] pathForResource: @"1557.b" ofType:
nil];

                if (bundlePath)
                {
                    [self.gnLookupLocalStream storageClear: &error];

                    if (! error)
                    {
                        __block GnLookupLocalStreamIngest *lookupLocalStreamIngest =
[[GnLookupLocalStreamIngest alloc] initWithGnLookupLocalStreamIngestEventsDelegate:self];

                        // Load Bundle in a separate thread to keep the UI responsive. This is
required for Large Bundles that can take few minutes to be ingested.

                        dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND,
0), ^{

                            NSError *error = nil;
                            NSInteger bytesRead = 0;
                            double totalBytesRead = 0;
                            uint8_t buffer[1024];
                            NSInputStream *fileInputStream = [NSInputStream
inputStreamWithFileAtPath:bundlePath];

                            [fileInputStream open];

                            do {
                                bytesRead =  [fileInputStream read:buffer maxLength:1024];
                                [lookupLocalStreamIngest write:buffer dataSize:sizeof(buffer)
error:&error];

                                if(error)
                                {
                                    NSLog(@"Error during lookupLocalStreamIngest write: %@", [error
localizedDescription]);
                                }

                                totalBytesRead+=bytesRead;

                            }while (bytesRead>0);
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
                            [lookupLocalStreamIngest flush:&error];
                            [fileInputStream close];

                        });
                    }
                }
            }
        }

        return error;
}
```

Windows Phone C#

```
// Ingest local fingerprint bundle
GnLookupLocalStream lookupLocalStream = GnLookupLocalStream.Enable;

lookupLocalStream.StorageLocation(Windows.Storage.ApplicationData.Current.LocalFolder.Path);

GnLookupLocalStreamIngest lookupLocalStreamIngest = new GnLookupLocalStreamIngest(this);

// Bundle assset path including asset name. The code assumes that the bundle
// to be ingested  was added to the project as a content file.
string bundlePath = "1557.b";

Stream bundleFileStream = Application.GetResourceStream(new Uri(bundlePath,
UriKind.Relative)).Stream;

byte[] buffer = new byte[1024 + 10];
int count = 1024;
int read = 0;
while (0 != (read = bundleFileStream.Read(buffer, 0, count)))
{
   lookupLocalStreamIngest.Write(buffer, (uint)read);
}

lookupLocalStreamIngest.Flush();
```

Designating a Storage Provider

A *storage provider* module implements GNSDK local storage. There are currently two available GNSDK storage providers: SQLite and QNX's QDB.

By default, no storage provider is enabled.To use SQLite, your application needs to instantiate a `GnStorageSqlite` object and call its `enable` method.

Identifying Music

The GNSDK supports identifying music in three different modules:

- **Music-ID**—Provides support for identifying music using the following:
  - CD TOC
  - Text search
  - Fingerprints
  - Gracenote identifier

© 2000 to present. Gracenote, Inc. All rights reserved.

The GnMusicId class supports this functionality. See "Identifying Music Using a CD-TOC, Text, or Fingerprints (MusicID)" for more information.

- **Music-ID File**—Provides support for identifying music stored in audio files. The GnMusicIdFile class supports this functionality. See "Identifying Audio Files (MusicID-File)" for more information.
- **Music-ID Stream**—Provides support for identifying music that is delivered in real-time as an end-user listens. For example: listening to a song on the radio or playing a song from a media player. You can identify streaming music using audio fingerprints generated from a streaming audio source, typically through a microphone. The GnMusicIdStream class provides support for this functionality. On some platforms, the GnMic class is available for listening to a device microphone.

  For more information, see "Identifying Streaming Audio (MusicID-Stream)".

Identifying Music Using a CD-TOC, Text, or Fingerprints (MusicID)

The MusicID module is the GNSDK component that handles recognition of non-streaming music though a CD TOC, audio source, fingerprint or search text. MusicID is implemented using the GnMusicId class. The MusicID-File module is the GNSDK component that handles audio file recognition, implemented through the GnMusicIdFile class. For information on identifying audio files and using the GnMusicIdFile class, see "Identifying Audio Files (MusicID-File)".

MusicID Queries

The GnMusicId class provides the following query methods:

- **FindAlbums**—Call this with an album or track identifier such as a CD TOC string, an audio source, a fingerprint, or identifying text (album title, track title, artist name, track artist name or composer name). This method returns a GnResponseAlbums object for each matching album.
- **FindMatches**—Call this method with identifying text. The method returns a GnResponseDataMatches object for each match, which could identify an album or a contributor.

Notes:

- A GnMusicId object's life time is scoped to a single recognition event and your application should create a new one for each event.
- During a recognition event, status events can be received via a delegate object that implements IGnStatusEvents (GnStatusEventsDelegate in Objective-C).
- A recognition event can be cancelled by the GnMusicId cancel method or by the "canceller" provided in each events delegate method.

Options for MusicID Queries

The GnMusicId::Options class allows you to set the following options:

© 2000 to present. Gracenote, Inc. All rights reserved.

- **LookupData**—Set `GnLookupData` options to enable what data can be returned, e.g., classical data, sonic data, playlist, external IDs, etc.
- **LookupMode**—Set a lookup option with one of the `GnLookupMode` enums. These include ones for local only, online only, online nocache, etc.
- **PreferResultLanguage**—Use one of the `GnLanguage` enums to set the preferred language for results.
- **PreferResultExternalId**—Set external ID for results from external provider. External IDs are 3rd party IDs used to cross link this metadata to 3rd party services.
- **PreferResultCoverart**—Specifies preference for results that have cover art associated.
- **ResultSingle**—Specifies whether a response must return only the single best result. Default is `true`.
- **ResultRangeStart**— Specifies result range start value.
- **ResultCount**— Specifies maximum number of returned results.

Identifying Music Using a CD TOC

MusicID-CD is the component of GNSDK that handles recognition of audio CDs and delivery of information including artist, title, and track names. The application provides GNSDK with the TOC from an audio CD and MusicID-CD will identify the CD and provide album and track information.

To identify music using a CD-TOC:

- Instantiate a `GnMusicId` object with your user handle.
- Set a string with your TOC values
- Call the `GnMusicId::FindAlbums` method with your TOC string.
- Process the `GnResponseAlbum` metadata result objects returned.

The code samples below illustrates a simple TOC lookup for local and online systems. The code for the local and online lookups is the same, except for two areas. If you are performing a local lookup, you must initialize the SQLite and Local Lookup libraries, in addition to the other GNSDK libraries:

C++ code sample

```
gnsdk_cstr_t toc= "150 14112 25007 41402 54705 69572 87335 98945 112902 131902 144055 157985 176900
189260 203342";
GnMusicId music_id(user);

music_id.Options().LookupData(kLookupDataContent);
GnResponseAlbums response = music_id.FindAlbums(toc);
```

Java and Android Java code sample

```
GnMusicId GnMusicId = new GnMusicId(user);
String toc = "150 14112 25007 41402 54705 69572 87335 98945 112902 131902 144055 157985 176900
189260 203342";
GnResponseAlbums responseAlbums = GnMusicId.findAlbums(toc);
```

Objective-C code sample

```
Under construction
```

C# and Windows Phone C# code sample

```
string toc = "150 14112 25007 41402 54705 69572 87335 98945 112902 131902 144055 157985 176900
189260 203342";
try
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
{
    using (GnStatusEventsDelegate midEvents = new MusicIdEvents())
    {
         GnMusicId GnMusicId = new GnMusicId(user, midEvents);
         GnResponseAlbums gnResponse = GnMusicId.FindAlbums(toc);
    }
}
```

Identifying Music Using Text

Using the GNSDK's MusicID module, your application can identify music using a lookup based on text strings. Besides user-inputted text, text strings can be extracted from an audio track's file path name and from text data embedded within a file, such as mp3 tags. You can provide the following types of input strings:

- Album title
- Track title
- Album artist
- Track artist
- Track composer

Text-based lookup attempts to match these attributes with known albums, artists, and composers. The text lookup first tries to match an album. If that is not possible, it next tries to match an artist. If that does not succeed, a composer match is tried. Adding as many input strings as possible to the lookup improves the results.

Text-based lookup returns "best-fit" objects, which means that depending on what your input text matched, you might get back album matches or contributor matches.

Identifying music using text is done using the `GnMusicId` class that has numerous methods for finding albums, tracks, and matches

To identify music using text:

1. Code an events handler object callbacks for status events (optional).
2. Instantiate a `GnMusicId` object with your User object and events handler object. Note that the events handler is optional.
3. Call the `GnMusicId::FindAlbums` method with your text search string(s).
4. Process metadata results returned

C++ code sample

```
/* Set the input text as album title, artist name, track title and perform the query */
GnResponseAlbums response = music_id.FindAlbums("Supernatural", "Africa Bamba", "Santana", GNSDK_
NULL, GNSDK_NULL);
```

Android Java and Java code sample

```
GnMusicId musicId = new GnMusicId( gnUser, new StatusEvents() );
GnResponseAlbums result = musicId.findAlbums( album, track, artist, null, null );
```

Objective-C code sample

```
musicId = [[GnMusicId alloc] initWithGnUser: self.gnUser statusEventsDelegate: self];
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
[self.cancellableObjects addObject: musicId];

[[musicId options] lookupData:kLookupDataContent bEnable:YES error:&amp;error];

self.queryBeginTimeInterval = [[NSDate date] timeIntervalSince1970];

[self enableOrDisableControls:NO];

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0), ^{
         NSError *textSearchOperationError = nil;
         GnResponseAlbums *responseAlbums = [musicId findAlbumsWithAlbumTitle: albumTitle
                                                                   trackTitle: trackTitle
                                                                   albumArtistName: artistName
                                                                   trackArtistName: artistName
                                                                   composerName: nil
                                                                   error: &amp;textSearchOperationError];
        });
}
```

C# and Windows Phone C# code sample

```
/* Set the input text as album title, artist name, track title and perfor the query */
GnResponseAlbums gnResponse = musicID.FindAlbums("Supernatural", "Africa Bamba", "Santana");
```

Identifying MusCic Using Fingerprints

You can identify music using an audio fingerprint. An audio fingerprint is data that uniquely identifies an audio track based on the audio waveform. You can use MusicID or MusicID-File to identify music using an audio fingerprint. The online Gracenote Media Service uses audio fingerprints to match the audio from a client application to the Gracenote Music Database. For more information, see *Fingerprint-Based Recognition*.

PCM Audio Format Recommendations

GNSDK fingerprinting supports the following PCM audio formats:

- **Sample Sizes**—16-bit
- **Channels**—1 or 2 (mono or stereo)
- **Sample Rates**—11025 Hz, 16000 Hz, 22050 Hz, 24000 Hz, 32000 Hz, 44100 Hz, 48000 Hz

Applications should use the highest quality audio possible to ensure the best results. Lower quality audio will result in less accurate fingerprint matches. Gracenote recommends at least 16-bit, stereo, 22050 Hz.

Do not resample or downsample audio to target these frequencies. Send the best quality audio that you have available.

MusicID Fingerprinting

The MusicID fingerprinting APIs give your application the ability to provide audio data as an identification mechanism. Note that if you want to do recognition using fingerprints and metadata together, and possibly have many files to do at once, then MusicID-File fingerprinting is probably the better solution. See *"Identifying Audio Files (MusicID-File)"*

© 2000 to present. Gracenote, Inc. All rights reserved.

There are four `GnMusicId` fingerprinting methods:

- **`FingerprintFromSource`**—Generates a fingerprint from a provided audio source. **Gracenote recommends using this**, as it encapsulates the below three calls (and additionally required code) into one.
- **`FingerprintBegin`**—Initialize fingerprint generation.
- **`FingerprintWrite`**—Provides uncompressed audio data for fingerprint generation. You can call this after `FingerprintBegin` to generate a native Gracenote Fingerprint Extraction (GNFPX) or Cantametrix (CMX) fingerprint.
- **`FingerprintEnd`**—Finalizes fingerprint generation.

Identifying music using MusicID fingerprinting examples:

C++ code sample

```
Under construction
```

Android Java code sample

```
Under construction
```

Objective-C code sample

```
Under construction
```

Windows Phone C# code sample

```
Under construction
```

C# code sample

```
/*---------------------------------------------------------------------------
*  SetFingerprintBeginWriteEnd
*/
private static void
SetFingerprintBeginWriteEnd(GnMusicId GnMusicId)
{
    bool complete = false;

    FileInfo file = new FileInfo(@"..\..\..\data\05-Hummingbird-sample.wav");

    using (BinaryReader b = new BinaryReader(File.Open(file.FullName, FileMode.Open,
FileAccess.Read)))
    {
                b.BaseStream.Position = 0;

                /* skip the wave header (first 44 bytes). we know the format of our sample files*/
                b.BaseStream.Seek(44, SeekOrigin.Begin);

                byte[] audioData = b.ReadBytes(2048);

                GnMusicId.FingerprintBegin(GnFingerprintType.kFingerprintTypeGNFPX, 44100, 16, 2);

                while (audioData.Length > 0)
                {
                    complete = GnMusicId.FingerprintWrite(audioData, (uint)audioData.Length);
                    if (true == complete)
                        break;
                    else
                        audioData = b.ReadBytes(2048);
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
                }

                GnMusicId.FingerprintEnd();

                if (false == complete)
                {
                    /* Fingerprinter doesn't have enough data to generate a fingerprint.
                        Note that the sample data does include one track that is too short to
fingerprint. */
                    Console.WriteLine("\nWarning: input file does contain enough data to generate a
fingerprint :" + file.FullName);
                }
    }
}

/*-------------------------------------------------------------------------
*  MusicidFingerprintAlbum
*/
private static void
MusicidFingerprintAlbum(GnUser user)
{
    Console.WriteLine("\n*****Sample MID-Stream Query*****");

    try
    {
        GnMusicId GnMusicId = new GnMusicId(user);

        /* Set the input fingerprint */
        SetFingerprintBeginWriteEnd(GnMusicId);

        /* Perform the search */
        GnResponseAlbums response = GnMusicId.FindAlbums(GnMusicId.FingerprintDataGet(),
GnFingerprintType.kFingerprintTypeGNFPX);

        DisplayFindAlbumResutlsByFingerprint(response);

    }
    catch (GnException e)
    {
    }
}
```

Java code sample

```
Under construction
```

Identifying Audio Files (MusicID-File)

The MusicID-File module can perform recognition using individual audio files or leverage collections of files to provide advanced recognition. When an application provides decoded audio and text data for each file to the library, MusicID-File identifies each file and, if requested, identifies groups of files as albums.

Music-ID File provides three mechanisms for identification:

1. **TrackID**—TrackID identifies the best album(s) for a single track. It returns results for an individual track independent of any other tracks submitted for processing at the same time.
2. **AlbumID**—AlbumID identifies the best album(s) for a group of tracks. Use AlbumID when

© 2000 to present. Gracenote, Inc. All rights reserved.

identifying submitted files as a group is important. For example, if all the submitted tracks were originally recorded on different albums, but exist together on a greatest hits album, then that will be the first match returned.

3. **LibraryID**—LibraryID identifies the best album(s) for a large collection of tracks. Besides metadata and other submitted tracks, LibraryID also takes into account a number of factors, such as location on device, when returning results.

For more information about the differences between TrackID, AlbumID, and LibraryID see the MusicID-File Overview

MusicID-File is implemented with the **`GnMusicIdFile`** class.

To identify audio from a file:

1. Code an `IGnMusicIdFileEvents` delegate class (`GnMusicIdFileEventsDelegate` in Objective-C) containing callbacks for results, events, and identification (metadata, fingerprint, etc.).
2. Instantiate a `GnMusicIdFile` object with your User object and events delegate object.
3. Call the `GnMusicIdFile::FileInfos` method to get a `GnMusicIdFileInfoManager` object.
4. For each file you want to identify, instantiate a `GnMusicIdFileInfo` object for it using the `GnMusicIdFileInfoManager` object's `Add` method.
5. For each instantiated `GnMusicIdFileInfo` object, set the file's path with the object's `FileName` method and assign it an identifier to correlate it with returned results.
6. Call one of the `GnMusicIdFile` query methods.
7. Handle metadata results in an events delegate callback.

Implementing an Events Delegate

To receive `GnMusicIdFile` notifications for results, events, and identification (metadata, fingerprint, etc.), your application needs to implement the `IGnMusicIdFileEvents` event delegate (`GnMusicIdFileEventsDelegate` in Objective-C) provided upon `GnMusicIdFile` object construction. This events delegate can contain callbacks for the following:

- Results handling
- Status event handling
- Other event handling
- Fingerprinting for identification
- Metadata for identification

Please note that these callbacks are optional, but you will want to code a results handling callback at a minimum, in order to get results. In addition, fingerprint and metadata identification can be done automatically when you create an audio file object (see next section) for each file you want to identify.

© 2000 to present. Gracenote, Inc. All rights reserved.

Android Java code sample

```
/**
* GNSDK MusicID-File event delegate
*/
private class MusicIDFileEvents extends IGnMusicIdFileEvents {

   HashMap<String, String> gnStatus_to_displayStatus;

   public MusicIDFileEvents(){
      gnStatus_to_displayStatus = new HashMap<String,String>();
      gnStatus_to_displayStatus.put("kMusicIdFileCallbackStatusProcessingBegin", "Begin processing
file");
      gnStatus_to_displayStatus.put("kMusicIdFileCallbackStatusFileInfoQuery", "Querying file
info");
      gnStatus_to_displayStatus.put("kMusicIdFileCallbackStatusProcessingComplete", "Identification
complete");
   }

   // ...other delegate events

}
```

Objective-C code sample

```
#pragma mark - MusicIDFileEventsDelegate Methods

-(void) musicIdFileAlbumResult: (GnResponseAlbums*)albumResult currentAlbum: (NSUInteger)
currentAlbum totalAlbums: (NSUInteger)totalAlbums cancellableDelegate: (id <GnCancellableDelegate>)
canceller
{
    [self.cancellableObjects removeObject: canceller];

    if (self.cancellableObjects.count==0)
    {
        self.cancelOperationsButton.enabled = NO;
    }

    [self processAlbumResponseAndUpdateResultsTable:albumResult];
}

    if(self.cancellableObjects.count==0)
    {
        self.cancelOperationsButton.enabled = NO;
    }

   [self enableOrDisableControls:YES];
   [self processAlbumResponseAndUpdateResultsTable:result];
}

   // ...other delegate events
```

Windows Phone C# code sample

```
#region IGnMusicIdFileEvents

      void IGnMusicIdFileEvents.GatherFingerprint(GnMusicIdFileInfo fileinfo, uint current_file,
uint total_files, IGnCancellable canceller)
      {
        return;
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
      }

      void IGnMusicIdFileEvents.GatherMetadata(GnMusicIdFileInfo fileinfo, uint current_file, uint
total_files, IGnCancellable canceller)
      {
        return;
      }

      void IGnMusicIdFileEvents.MusicIdFileComplete(GnError musicidfile_complete_error)
      {
            List<AlphaKeyGroup<RespAlbum>> DataSource = AlphaKeyGroup<RespAlbum>.CreateGroups
(respAlbList_,
            System.Threading.Thread.CurrentThread.CurrentUICulture,
            (RespAlbum s) => { return s.Title; }, true);

              Deployment.Current.Dispatcher.BeginInvoke(() =>
              {
              TBStatus.Text = "Status : MusicIDFile Completed Successfully";
                ToggleUIBtnsVisibility(true);
              LLRespAlbum.ItemsSource = DataSource;
              });
      }

      // ...other delegate events
```

C++ code sample

```
/*
 * Callback delegate classes
 */

/* Callback delegate called when performing MusicID-File operation */
class MusicIDFileEvents : public IGnMusicIdFileEvents
{
public:
       virtual void
       StatusEvent(GnStatus status,
                                      gnsdk_uint32_t percent_complete,
                                      gnsdk_size_t bytes_total_sent,
                                      gnsdk_size_t bytes_total_received,
                                      IGnCancellable& canceller)
       {
               std::cout << "status (";

               switch (status)
               {
               case gnsdk_status_unknown:
                       std::cout <<"Unknown ";
                       break;

               case gnsdk_status_begin:
                       std::cout <<"Begin ";
                       break;

               case gnsdk_status_connecting:
                       std::cout <<"Connecting ";
                       break;

               case gnsdk_status_sending:
                       std::cout <<"Sending ";
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
                        break;

                case gnsdk_status_receiving:
                        std::cout <<"Receiving ";
                        break;

                case gnsdk_status_disconnected:
                        std::cout <<"Disconnected ";
                        break;

                case gnsdk_status_complete:
                        std::cout <<"Complete ";
                        break;

                default:
                        break;
                }

                std::cout << "), % complete ("
                                        << percent_complete
                                        << "), sent ("
                                        << bytes_total_sent
                                        << "), received ("
                                        << bytes_total_received
                                        << ")"
                                        << std::endl;

                GNSDK_UNUSED(canceller);
        }
    //... more delegate events


};
```

C# code sample

```
Under construction
```

Java code sample

```
Under construction
```

Adding Audio Files for Identification

To add audio files for identification:

1. Call the `GnMusicIdFile::FileInfos` method to get a `GnMusicIdFileInfoManager` object.
2. For each file you want to identify, instantiate a `GnMusicIdFileInfo` object for it using the `GnMusicIdFileInfoManager` object's `Add` method.

   Each audio file must be added with a unique identifer string that your application can use to correlate results in callbacks with a specific file. Audio files can be added as a `GnAudioFile` (only available on some platforms) instance.

© 2000 to present. Gracenote, Inc. All rights reserved.

3. Set the file's path with the `GnMusicIdFileInfo::FileName` method.

   `GnMusicIdFileInfo` objects are used to contain the metadata that will be used in identification and will also contain results after a query has completed. MusicID-File matches each `GnMusicIdFileInfo` object to a track within an album.

   Adding audio files as a `GnAudioFile` instance (only available on some platforms) allows `GnMusicIdFile` to automatically extract metadata available in audio tags (artist, album, track, track number, etc.,) and fingerprint the raw audio data, saving your application the need to do this in event delegate methods.

Setting Audio File Identification

To aid in identification, your application can call `GnMusicIdFileInfo` methods to get/set CDDB IDs, fingerprint, path and filename, FileInfo identifier (set when FileInfo created), media ID (from Gracenote), source filename (from parsing) and application, Media Unique ID (MUI), Tag ID (aka Product ID), TOC offsets, track artist/number/title, and TUI (Title Unique Identifier).

Android Java code sample

```
@Override
public void gatherMetadata(GnMusicIdFileInfo fileInfo, long currentFile, long totalFiles,
IGnCancellable cancelable) {

   MediaMetadataRetriever mmr = new MediaMetadataRetriever();
   try {
          mmr.setDataSource(fileInfo.fileName());
          fileInfo.albumTitle(mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ALBUM));
          fileInfo.albumArtist(mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_
ALBUMARTIST));
          fileInfo.trackTitle(mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_TITLE));
          fileInfo.trackArtist(mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ARTIST));
          try {
                  long trackNumber = Long.parseLong(mmr.extractMetadata
(MediaMetadataRetriever.METADATA_KEY_CD_TRACK_NUMBER));
                  fileInfo.trackNumber(trackNumber);
                } catch (NumberFormatException e) {}

           } catch (IllegalArgumentException e1) {
                 Log.e(appString, "illegal argument to MediaMetadataRetriever.setDataSource");
        } catch (GnException e1) {
                 Log.e(appString, "error retrieving filename from fileInfo");
           }
}
```

Objective-C code sample

```
-(void) gatherMetadata: (GnMusicIdFileInfo*) fileInfo
                  currentFile: (NSUInteger) currentFile
                       totalFiles: (NSUInteger) totalFiles
   cancellableDelegate: (id <GnCancellableDelegate>) canceller
{
    NSError *error = nil;
    NSString* filePath = [fileInfo fileName:&error];

    if (error)
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
    {
        NSLog(@"Error while retrieving filename %@ ", [error localizedDescription]);
    }
    else
    {
        AVAsset *asset = [AVAsset assetWithURL:[NSURL fileURLWithPath:filePath]];
        if (asset)
        {
            NSArray *metadataArray =  [asset metadataForFormat:AVMetadataFormatID3Metadata];

            for(AVMetadataItem* item in metadataArray)
            {
                NSLog(@"AVMetadataItem Key = %@ Value = %@",item.key, item.value );

                if([[item commonKey] isEqualToString:@"title"])
                {
                    [fileInfo trackTitleWithValue:(NSString*) [item value] error:nil];
                }
                else if([[item commonKey] isEqualToString:@"albumName"])
                {
                    [fileInfo albumTitleWithValue:(NSString*) [item value] error:nil];
                }
                else if([[item commonKey] isEqualToString:@"artist"])
                {
                    [fileInfo trackArtistWithValue:(NSString*) [item value] error:nil];
                }
            }
        }
    }
}
```

Windows Phone C# code sample

```
void IGnMusicIdFileEvents.GatherMetadata(GnMusicIdFileInfo fileinfo, uint current_file, uint total_
files, IGnCancellable canceller)
{
   return;
}
```

C++ code sample

```
GnMusicIdFileInfo fileinfo;
fileinfo = midf.FileInfos().Add(fileIdent);
fileinfo.FileName(filePath);
fileInfo.AlbumArtist( "kardinal offishall" );
fileInfo.AlbumTitle ( "quest for fire" );
fileInfo.TrackTitle ( "intro" );
```

C# code sample

```
Under construction
```

Java code sample

```
Under construction
```

© 2000 to present. Gracenote, Inc. All rights reserved.

MusicID-File Fingerprinting

The MusicID-File fingerprinting APIs give your application the ability to provide audio data as an identification mechanism. This enables MusicID-File to perform identification based on the audio itself, as opposed to performing identification using only the associated metadata. Use the MusicID-File fingerprinting APIs during an events delegate callback.

There are four `GnMusicIdFileInfo` fingerprinting methods:

- **`FingerprintFromSource`**—Generates a fingerprint from a provided audio source. **Gracenote recommends using this**, as it encapsulates the below three calls (and additionally required code) into one.
- **`FingerprintBegin`**—Initialize fingerprint generation.
- **`FingerprintWrite`**—Provides uncompressed audio data for fingerprint generation.
- **`FingerprintEnd`**—Finalizes fingerprint generation.

Android Java code sample

```
@Override
public void gatherFingerprint(GnMusicIdFileInfo fileInfo, long currentFile, long totalFiles,
IGnCancellable cancelable){
   try {
          fileInfo.fingerprintFromSource( new GnAudioFile( new File(fileInfo.fileName())) );
       } catch (GnException e) {
         Log.e(appString, "error in fingerprinting file: " + e.getErrorAPI() + ", " +
e.getErrorModule() + ", " + e.getErrorDescription());
       }
}
```

Objective-C code sample

```
[musicIDFileInfo fingerprintFromSource:(id <GnAudioSourceDelegate> )gnAudioFile error:&error];
```

Windows Phone C# code sample

```
Under construction
```

C++ code sample

```
file = fileInfo.FileName();

std::ifstream audioFile (file, std::ios::in | std::ios::binary);
if ( audioFile.is_open() )
{
    /* skip the wave header (first 44 bytes). the format of the sample files is known,
     * but please be aware that many wav file headers are larger then 44 bytes!
     */
    audioFile.seekg(44);
    if ( audioFile.good() )
    {
        /* initialize the fingerprinter
         * Note: The sample files are non-standard 11025 Hz 16-bit mono to save on file size
         */
         fileInfo.FingerprintBegin(11025, 16, 1);

        do
        {
            audioFile.read(pcmAudio, 2048);
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
            complete = fileInfo.FingerprintWrite((gnsdk_byte_t*)pcmAudio,
                                                 (gnsdk_size_t)audioFile.gcount()
                    );

            /* does the fingerprinter have enough audio? */
            if (GNSDK_TRUE == complete)
            {
                break;
            }
        }
        while ( audioFile.good() );

       if (GNSDK_TRUE != complete)
       {
            /* Fingerprinter doesn't have enough data to generate a fingerprint.
               Note that the sample data does include one track that is too short to fingerprint. */
           std::cout << "Warning: input file does contain enough data to generate a fingerprint:\n"
<< file <<"\n";
           fileInfo.FingerprintEnd();
        }
    }
    else
    {
        std::cout << "\n\nError: Failed to skip wav file header: " << file <<"\n\n";
    }
}
```

C# code sample

```
/*-------------------------------------------------------------------------
 *  GatherFingerprint
 */
public override void
GatherFingerprint(GnMusicIdFileInfo fileInfo, uint currentFile, uint totalFiles, IGnCancellable
canceller)
{
    byte[]     audioData  = new byte[2048];
    bool       complete   = false;
    int        numRead    = 0;
    FileStream fileStream = null;

    try
    {
        string filename = fileInfo.FileName;
        if (filename.Contains('\\'))
            fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);
        else
            fileStream = new FileStream(folderPath + filename, FileMode.Open, FileAccess.Read);

       /* check file for existence */
       if (fileStream == null || !fileStream.CanRead)
       {
           Console.WriteLine("\n\nError: Failed to open input file: " + filename);
       }
       else
       {
          /* skip the wave header (first 44 bytes). we know the format of our sample files, but
please
             be aware that many wav file headers are larger then 44 bytes! */
           if (44 != fileStream.Seek(44, SeekOrigin.Begin))
           {
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
                Console.WriteLine("\n\nError: Failed to seek past header: %s\n", filename);
            }
            else
            {
                /* initialize the fingerprinter
                   Note: Our sample files are non-standard 11025 Hz 16-bit mono to save on file size
*/
                fileInfo.FingerprintBegin(11025, 16, 1);

                numRead = fileStream.Read(audioData, 0, 2048);
                while ((numRead) > 0)
                {
                    /* write audio to the fingerprinter */
                    complete = fileInfo.FingerprintWrite(audioData, Convert.ToUInt32(numRead));

                   /* does the fingerprinter have enough audio? */
                    if (complete)
                    {
                        break;
                    }

                    numRead = fileStream.Read(audioData, 0, 2048);
                }
                fileStream.Close();

                /* signal that we are done */
                fileInfo.FingerprintEnd();
                Debug.WriteLine("Fingerprint: " + fileInfo.Fingerprint + " File: " +
fileInfo.FileName);
            }
        }

        if (!complete)
        {
            /* Fingerprinter doesn't have enough data to generate a fingerprint.
            Note that the sample data does include one track that is too short to fingerprint. */
           Console.WriteLine("Warning: input file does contain enough data to generate a
fingerprint:\n" + filename);
        }

    }
    catch (FileNotFoundException e)
    {
        Console.WriteLine("FileNotFoundException " + e.Message);
    }
    catch (IOException e)
    {
        Console.WriteLine("IOException " + e.Message);
    }
    finally
    {
        try
        {
            fileStream.Close();
        }
        catch (IOException e)
        {
            Console.WriteLine("IOException " + e.Message);
        }
    }
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
}
```

Java code sample

```
Under construction
```

Setting Options for MusicID-File Queries

To set an option for your MusicID-File query, instantiate a `GnMusicIdFileOptions` object using the `GnMusicIdFile::Options` method and call its methods. For example, you can set an option for local lookup. By default, a lookup is handled online, but many applications will want to start with a local query first then, if no match is returned, fall back to an online query.

MusicID-File Query Options:

- **`LookupData`**—Set `GnLookupData` options to enable what data can be returned, e.g., classical data, sonic data, playlist, external IDs, etc.
- **`LookupMode`**—Set a lookup option with one of the `GnLookupMode` enums. These include ones for local only, online only, online nocache, etc.
- **`BatchSize`**— In LibraryID, you can set the batch size to control how many files are processed at a time. The higher the size, the more memory will be used. The lower the size, the less memory will be used and the faster results will be returned.
- **`ThreadPriority`**—Use one of the `GnThreadPriority` enums to set thread priority, e.g., default, low, normal, high, etc.
- **`OnlineProcessing`**—Enable (`true`) or disable (`false`) online processing.
- **`PreferResultLanguage`**—Use one of the `GnLanguage` enums to set the preferred language for results.
- **`PreferResultExternalId`**—Set external ID for results from external provider. External IDs are 3rd party IDs used to cross link this metadata to 3rd party services.

Making a MusicID-File Query

`GnMusicIdFile` provides the following query methods:

- **`DoTrackId`**—Perform a Track ID query.
- **`DoTrackIdAsync`**—Perform an asynchronous Track ID query.
- **`DoAlbumId`**—Perform an Album ID query.
- **`DoAlbumIdAsync`**—Perform an asynchronous Album ID query.
- **`DoLibraryId`**—Perform a Library ID query.
- **`DoLibraryIdAsync`**—Perform an asynchronous Library ID query.

Note that GNSDK processing is always done asynchronously and returns results via callbacks. With the blocking functions, your app waits until processing has completed before continuining.

Options When Making Query Call

When you make a `GnMusicIdFile` query method call, you can set the following options at the time of the call (as opposed to setting options with `GnMusicIdFileOptions` object—see above):

© 2000 to present. Gracenote, Inc. All rights reserved.

- **Return matches** - Have MusicID-File return all results found for each given `GnMusicIdFileInfo`
- **Return albums** - Only album matches are returned (default)
- **Return all** - Have MusicID-File return all results found for each given `GnMusicIdFileInfo`
- **Return single** - Have MusicID-File return the single best result for each given `GnMusicIdFileInfo` (default)

C++ code sample

```
/* Launch AlbumID */
midf.DoAlbumId(kQueryReturnSingle, kResponseAlbums );
```

Objective-C code sample

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),  ^{
   NSError *error = nil;
   [musicIDFileInfo fingerprintFromSource:(id <GnAudioSourceDelegate> )gnAudioFile error:&error];
   error = nil;
   [gnMusicIDFile doAlbumId:kQueryReturnSingle responseType:kResponseAlbums error:&error];
});
```

Identifying Streaming Audio (MusicID-Stream)

The functionality for identifying streaming audio is contained in the **Music-ID Stream** module. This module contains the `GnMusicIdStream` class that is designed to identify raw audio received in a continuous stream. You should instantiate one instance of this class for each audio stream you are using.

Using this class, your application primarily needs to provide two things:

1. Code an `IGnMusicIdStreamEvents` events delegate object (`GnMusicIdStreamEventsDelegate` in Objective-C) that receives callbacks for results, status messages, and other events.
2. Code a class that implements the `IGnAudioSource` (`GnAudioSource` in Objective-C) interface.

For some platforms (e.g., iOS and Android), Gracenote provides the `GnMic` helper class that implements the `IGnAudioSource` interface. If available, your application can use this class to process streaming audio from a device microphone.

Notes:

- At any point, your application can request an identification of buffered audio. The identification process spawns a thread and completes asynchronously. Use the method `IdentifyAsync` to identify music.
- Audio is identified using either a local database or the Gracenote online service. The default is to attempt a local identification first before going online. Local matches are only possible if `GnLookupLocalStream` is enabled and a MusicID-Stream fingerprint bundle ingested.
- You can call `IdentifyCancel` to stop an identification operation. Note that cancelling does not cease audio processing and your application can continue requesting identifications.
- At any point, your application can stop audio processing. When stopped, automatic data fetching

© 2000 to present. Gracenote, Inc. All rights reserved.

ceases or, if audio data is being provided manually, attempts to write data for processing will fail. Internally, `GnMusicIdStream` clears and releases all buffers and audio processing modules. Audio processing can be restarted at any time.

- You can instantiate a `GnMusicIdStream` object with a locale. Locales are a convenient way to group locale-dependent metadata specific to a region (such as Europe) and language that should be returned from the Gracenote service. See "Loading a Locale" for more information.

To identify streaming audio from an audio source (`IGnAudioSource` implementation):

1. Code a `IGnMusicIdStreamEvents` delegate class (`GnMusicidStreamEventsDelegate` in Objective-C) to handle results, statuses, and other events.
2. Instantiate a `GnMusicIdStream` object with your User object and events delegate object. This establishes a MusicID-Steam audio channel.
3. Instantiate a `GnMic` object and call its `getData` method to retrieve PCM audio data.
4. Call the `GnMusicIdStream.audioProcess` method to buffer your captured PCM data.
5. When the user requests an identification, call the `GnMusicIdStream` instance's `identifyAlbumAsync` method.
6. Instantiate an audio source object representing the audio source you wish to identify. On some platforms, Gracenote provides the `GnMic` class, which is an `IGnAudioSource` implementation for the device mircophone.
7. Call the `GnMusicIdStream` instance's `AudioProcessStart` method with your `IGnAudioSource` object. This starts the retrieval and processing of audio.

   Note that, as an alternative to the above two steps, you could call the `GnMusicIdStream` instance's `audioProcess` method with PCM string data that you have captured through any means.

8. To identify audio, call `GnMusicIdStream.IdentifyAlbumAsync`. Results are delivered asychnronously as audio is received.
9. Handle results and statuses in your event delegate callbacks.

Examples

Android Java code sample

```
// Initialize mic
gnMicrophone = new GnMic(44100, 16, 1);
gnMicrophone.sourceInit();

// Initialize music id stream
gnMusicIdStream = new GnMusicIdStream(gnMusicUser, new GnMusicIdStreamEvents());
gnMusicIdStream.audioProcessStart(
      gnMicrophone.samplesPerSecond(),
      gnMicrophone.sampleSizeInBits(),
      gnMicrophone.numberOfChannels());

// Create thread to process audio data from the mic
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
Thread audioProcessThread = new Thread(new Runnable()
{
   @Override
   public void run()
   {
      try
      {
         ByteBuffer byteBuffer = ByteBuffer.allocateDirect(1024*4);
         long bytesRead = 0;
         while(isListening)
         {
            bytesRead = gnMicrophone.getData(byteBuffer, byteBuffer.capacity());
            gnMusicIdStream.audioProcess(byteBuffer.array(), bytesRead);
         }
      }
   }
});

// Call this method when user requests identification
 gnMusicIdStream.identifyAlbumAsync();

// Handle results in callback
```

Objective-C code sample

```
mMic = [[GnMic alloc] initWithSampleRate:44100 bitsPerChannel:8 numberOfChannels:2];
mMusicAudio = [[GnMusicIdStream alloc] initWithGnUser:[self getUserMusic]
musicIdStreamEventsDelegate: self];
-(void)start
{
    NSError *error = nil;

    if(mMic & (0 == [mMic sourceInit]))
    {
       if (mMusicAudio)
       {
           [mMusicAudio audioProcessStartWithSamplesPerSecond:[mMic samplesPerSecond] bitsPerSample:
[mMic sampleSizeInBits] numberOfChannels:[mMic numberOfChannels] error:&error];
       }

    }
   [self performSelectorInBackground:@selector(processAudio) withObject:nil];
}

-(void)processAudio
{
    NSError *error = nil;
    NSMutableData *data = [[NSMutableData alloc] initWithLength:1024];
    while (true)
    {
        NSInteger length = [mMic getData:data];
        if(data)
        {
            [mMusicAudio audioProcess:data error:&error];
        }
    }
}

//Call this when user request identification
 [mMusicAudio identifyAlbumAsync:&error];
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
// Handle results in callback
```

Windows Phone C# code sample

```
gnMic = new GnMic();
gnMic.SourceInit();
App.gnMusicIDStream_ = new GnMusicIdStream(App.gnMusicUser_, this);
if (gnMic.SampleSizeInBits() == 16)
{
   App.gnMusicIDStream_.AudioProcessStart(gnMic.SamplesPerSecond(), 16, gnMic.NumberOfChannels());
}
else
{
   App.gnMusicIDStream_.AudioProcessStart(gnMic.SamplesPerSecond(), 8, gnMic.NumberOfChannels());
}
//...
bytesRead = gnMic.GetData(audioData, (uint)audioData.Length);
App.gnMusicIDStream_.AudioProcess(audioData, bytesRead);

// Call this when user requests identification
App.gnMusicIDStream_.IdentifyAlbumAsync();

//Handle results in callback
```

Setting Options for Streaming Audio Queries

You can use `GnMusicIdStreamOptions` methods to set options for streaming audio queries. For example, you can set an option for local lookup. By default, a lookup is done online, but many applications will want to start with a local query first then, if no match is found, go online.

`GnMusicIdStreamOptions` Query Option Methods:

- **`LookupData`**—Set `GnLookupData` options to enable what data can be returned, e.g., classical data, sonic data, playlist, external IDs, etc.
- **`LookupMode`**—Set a lookup option with one of the `GnLookupMode` enums. These include ones for local only, online only, online nocache, etc.
- **`PreferResultLanguage`**—Use one of the `GnLanguage` enums to set the preferred language for results.
- **`PreferResultExternalId`**—Specifies preference for results with external IDs. External IDs are 3rd party IDs used to cross link this metadata to 3rd party services.
- **`PreferResultCoverart`**—Specifies preference for results with cover art.
- **`ResultSingle`**—Specifies whether a response must return only the single best result. Default is `true`.
- **`ResultRangeStart`**— Specifies the result range start value. This must be less than or equal to the total number of results. If greater than the total number, no results are returned.
- **`ResultCount`**— Specifies the result range count value.
- **`RevisionCheck`**— Request that a revision check be performed.

Music-ID Stream Code Samples

Android Java code sample

```
MusicIDStreamEvents musicIDStreamEvents = new MusicIDStreamEvents();
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
/**
 * GNSDK MusicID-Stream event delegate
 */
private class MusicIDStreamEvents extends GnMusicIdStreamEventsListener {

    HashMap<String, String> gnStatus_to_displayStatus;

    public MusicIDStreamEvents(){
        gnStatus_to_displayStatus = new HashMap<String,String>();
        gnStatus_to_displayStatus.put("kStatusStarted", "Identification started");
        gnStatus_to_displayStatus.put("kStatusFpGenerated", "Fingerprinting complete");

        //gnStatus_to_displayStatus.put("kStatusIdentifyingOnlineQueryStarted", "Online query
started");
        gnStatus_to_displayStatus.put("kStatusIdentifyingEnded", "Identification complete");
    }

    @Override
    public void statusEvent( GnStatus status, long percentComplete, long bytesTotalSent, long
bytesTotalReceived, IGnCancellable cancellable ) {
        //setStatus( String.format("%d%%",percentComplete), true );
    }

    @Override
    public void musicIdStreamStatusEvent( GnMusicIdStreamStatus status, IGnCancellable cancellable )
{
        if(gnStatus_to_displayStatus.containsKey(status.toString())){
             setStatus( String.format("%s", gnStatus_to_displayStatus.get(status.toString())), true
);
        }
    }

    @Override
    public void musicIdStreamResultAvailable( GnResponseAlbums result, IGnCancellable cancellable )
{
        activity.runOnUiThread(new UpdateResultsRunnable( result ));
        setStatus( "Success", true );
        setUIState( UIState.READY );
    }
}

private void startContinuousListening() throws GnException{

    if ( gnMicrophone == null ){
        gnMicrophone = new GnMic();
        gnMicrophone.start();
    } else {
        gnMicrophone.resume();
    }

    GnMusicIdStream = new GnMusicIdStream( gnUser, musicIDStreamEvents );
    queryObjects.add( GnMusicIdStream.canceller() ); // retain event object so we can cancel if
requested
    GnMusicIdStream.options().resultSingle( true );

    Thread audioProcessThread = new Thread(new AudioProcessRunnable());
    audioProcessThread.start();
}

/**
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
 * GnMusicIdStream object processes audio read directly from GnMic object
 */
class AudioProcessRunnable implements Runnable {

    @Override
    public void run() {
        try {

                GnMusicIdStream.audioProcessStart( gnMicrophone );

        } catch (GnException e) {
            Log.e( appString, e.getErrorCode() + ", " + e.getErrorDescription() + ", " +
e.getErrorModule() );
            showError( e.getErrorAPI() + ": " +  e.getErrorDescription() );

        }
    }
}
```

Objective-C code sample

```
-(void) setupMusicIDStream
{
    __block NSError *musicIDStreamError = nil;
    self.gnMusicIDStream = [[GnMusicIDStream alloc] initWithUser:self.gnUser
musicIDStreamEventsDelegate:self error:&musicIDStreamError];

    musicIDStreamError = nil;
    [self.gnMusicIDStream optionResultSingle:YES error:&musicIDStreamError];

    musicIDStreamError = nil;
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^
       {
           [self.gnMusicIDStream audioProcessStartWithAudioSource:self.gnMic
error:&musicIDStreamError];

           if(musicIDStreamError)
           {
               dispatch_async(dispatch_get_main_queue(), ^
               {

                   NSLog(@"Error while starting Audio Process With AudioSource - %@",
[musicIDStreamError localizedDescription]);
                });
           }

       });
}
#pragma mark - GnMusicIDStreamEventsDelegate Methods

-(void) midstreamStatusEvent:(GnMusicIDStream*) gnMusicIDStream status:(GnAudioStatus) status
{
    NSString *statusString = nil;

    switch (status)
    {
        case  AudioStatusInvalid:
            statusString = @"Error";
            [self performSelectorOnMainThread:@selector(updateStatus:) withObject:statusString
waitUntilDone:NO];
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
            break;
        case AudioStatusIdentifyingStarted:
            statusString = @"Identifying";
            [self performSelectorOnMainThread:@selector(updateStatus:) withObject:statusString
waitUntilDone:NO];
            break;

        /*... More audio statuses */

    }
}
```

Windows Phone C# code sample

```
// Create and setup gracenote musicid stream instance
App.mMusicIDStream = new GnMusicIdStream(App.mGnUser, this);
App.mMusicIDStream.Options.ResultSingle(true);
App.mMusicIDStream.Options.LookupData(GnLookupData.kLookupDataContent,true);
App.mDispatcherTimer.Start();
App.mMicrophone.Start();
App.mMusicIDStream.AudioProcessStart((uint)App.mMicrophone.SampleRate, 16, 1);

#region IGnMusicIdStreamEvents

void IGnMusicIdStreamEvents.MusicIdStreamResultAvailable(GnResponseAlbums result, IGnCancellable
canceller)
{
   ClearResults();
   ShowAlbums(result, true, false);
}

void IGnMusicIdStreamEvents.MusicIdStreamStatusEvent(GnMusicIdStreamStatus status_, IGnCancellable
canceller)
{
   return;
}

void IGnMusicIdStreamEvents.StatusEvent(GnStatus status_, uint percent_complete, uint bytes_total_
sent, uint bytes_total_received, IGnCancellable canceller)
{
   Deployment.Current.Dispatcher.BeginInvoke(() =>
   {
      string status = null;
      switch (status_)
      {
         case GnStatus.kStatusDisconnected: status = "Disconnected"; break;
         case GnStatus.kStatusBegin:     status = "Begin";          break;
         case GnStatus.kStatusComplete:  status = "Complete";       break;
         case GnStatus.kStatusConnecting:status = "Connecting";     break;
         case GnStatus.kStatusProgress:  status = "Progress";       break;
         case GnStatus.kStatusReading:   status = "Reading";        break;
         case GnStatus.kStatusSending:   status = "Sending";        break;
         case GnStatus.kStatusUnknown:   status = "Unknown";        break;
         case GnStatus.kStatusWriting:   status = "Writing";        break;
         case GnStatus.kStatusErrorInfo: status = "ErrorInfo";      break;

      }

       TBStatus.Text = "Status : " + status + "\t(" + percent_complete.ToString() + "%)";
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
     });

 }
#endregion
```

C++ code sample

```
/*-------------------------------------------------------------------------
*  do_musicid_stream
*/
static void
do_musicid_stream(GnUser& user)
{
   std::cout << std::endl << "*****Sample MID-Stream Query*****" << std::endl;

   MusicIDStreamEvents* mids_events = new MusicIDStreamEvents;

   try
   {
      GnMusicIdStream mids(user, mids_events);

      /* create microphone to use as audio source */
      GnMic        mic(44100, 16, 1);
      /* pass Mic through GnWavCapture to record audio for testing (optional) */
      GnWavCapture wavrec(mic, "mic_record.wav");

      /* We want to add Content (eg: image URLs) data to our lookups */
      mids.Options().LookupData(kLookupDataContent, true);

      /* starts identification in background thread - will wait for audio if none yet */
      mids.IdentifyAsync();

      /* provide audio continuously until GnMusicIdStream::AudioProcessStop() is called */
      mids.AudioProcessStart(wavrec);

      /* We called GnMusicIdStream::AudioProcessStop() on 'gnsdk_mids_identifying_ended'
       * status callback which caused the above to return */
   }
   catch (GnError& e)
   {
      std::cout << e.ErrorAPI() << "\t" << std::hex << e.ErrorCode() << "\t" >>  e.ErrorDescription
() << std::endl;
   }

   delete mids_events;
}

/* IGnStatusEvents : overrider methods of this class to get delegate callbacks */
class MusicIDStreamEvents : public IGnMusicIdStreamEvents
{
   /*-------------------------------------------------------------------------
    *  StatusEvent
    */
   void
   StatusEvent(gracenote::GnStatus status, gnsdk_uint32_t percent_complete, gnsdk_size_t bytes_
total_sent, gnsdk_size_t bytes_total_received, IGnCancellable& canceller)
   {
      std::cout << "status (";

      switch (status)
      {
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
        case gnsdk_status_unknown:
             std::cout <<"Unknown ";
             break;

        case gnsdk_status_begin:
             std::cout <<"Begin ";
             break;

        //...more statuses
     }
   }

   /*-----------------------------------------------------------------------------
    *  MidStreamStatusEvent
    */
   virtual void
      MusicIdStreamStatusEvent(GnMusicIdStreamStatus status, IGnCancellable& canceller)
      {
         switch (status)
         {
            case gnsdk_mids_status_invalid:
            std::cout <<"Invalid status!" << std::endl;
            break;

            case gnsdk_mids_identifying_started:
            std::cout <<"Identification: started" << std::endl;
                      break;

                      //...more statuses

         }
      }

/*-----------------------------------------------------------------------------
 *  MidStreamResultAvailable
 */
virtual void
    MusicIdStreamResultAvailable(GnResponseAlbums& result, IGnCancellable& canceller)
    {
       if (result.ResultCount() > 0)
       {
          metadata::album_iterator it_album = result.Albums().begin();

          for (; it_album != result.Albums().end(); ++it_album)
          {
             GnAlbum album = *it_album;

             /* Get Track Artist, if not available, use Album Artist */
             gnsdk_cstr_t artist_name = album.TracksMatched()[0]->Artist().Name().Display();
             if (artist_name == GNSDK_NULL)
                artist_name = album.Artist().Name().Display();

             /* Get cover art URL for 'small' image */
             gnsdk_cstr_t cover_url = album.Content(kContentTypeImageCover).Asset
(kImageSizeSmall).Url();
             if (cover_url == GNSDK_NULL)
                cover_url = "no url";

             std::cout <lt; std::endl;
             std::cout <lt; "      Track : " <lt; album.TracksMatched()[0]->Title().Display() <lt;
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
std::endl;
             std::cout <lt; "      Artist: " <lt; artist_name <lt; std::endl;
             std::cout <lt; "      Cover : " <lt; cover_url <lt; std::endl;
          }
       }
       else
       {
          std::cout <lt; "      No match found" <lt; std::endl;
       }
       std::cout <lt; std::endl;

       GNSDK_UNUSED(canceller);
    }
};
```

Processing Returned Metadata Results

Processing returned metadata results is pretty straight-forward—objects returned can be traversed and accessed like any other objects. However, there are three things about returned results you need to be aware of:

1. **Needs decision**—A result could require an additional decision from an application or end user.
2. **Full or partial results**—A result object could contain full or partial metadata
3. **Locale-dependent data**—Some metadata requires that a locale be loaded.

Needs Decision

Top-level response classes—`GnResponseAlbums`, `GnResponseTracks`, `GnResponseContributors`, `GnResponseDataMatches`—contain a "needs decision" flag. In all cases, Gracenote returns high-confidence results based on the identification criteria; however, even high-confidence results could require additional decisions from an application or end user. For example, all multiple match responses require the application (or end user) make a decision about which match to use and process to retrieve its metadata. Therefore, the GNSDK flags every multiple match response as "needs decision".

A single match response can also need a decision. Though it may be a good candidate, Gracenote could determine that it is not quite perfect.

In summary, responses that require a decision are:

- Every multiple match response
- A single match response that Gracenote determines needs a decision from the application or end user, based on the quality of the match and/or the mechanism used to identify the match (such as text, TOC, fingerprints, and so on).

Example needs decision check (C++):

```
GnResponseAlbums response = music_id.FindAlbums(albObject);
needs_decision =response.NeedsDecision();
if(needs_decision)
{
    /* Get user selection. Note that is not an SDK method */
       user_pick = doMatchSelection(response);
}
```

© 2000 to present. Gracenote, Inc. All rights reserved.

Objective-C:

```
/* Perform the query */
GnResponseAlbums *response = [musicId findAlbumsWithGnDataObject:dataAlbum error:&error];

if ( [[response albums] allObjects].count != 0)
{
   needsDecision = [response needsDecision];

   /* See if selection of one of the albums needs to happen */
   if (needsDecision)
   {
      // Get User selection. Note that is not an SDK method
         choiceOrdinal = [self doMatchSelection:response];
   }
   // ...

}
```

Full and Partial Metadata Results

A query response can return 0-n matches. As indicated with a `fullResult` flag, a match can contain either full or partial metadata results. A partial result is a subset of the full metadata available, but enough to perform additional processing. One common use case is to present a subset of the partial results (e.g., album titles) to the end user to make a selection. Once a selection is made, you can then do a secondary query, using the partial object as a parameter, to get the full results (if desired).

Example followup query (C++):

```
/* Get first match */
GnAlbum album = response.Albums().at(0).next();
bool fullResult = album.IsFullResult();

/* If partial result, do a follow-up query to retrieve full result */
if (!fullResult)
{
    /* Do followup query to get full data - set partial album as query input. */
    GnResponseAlbums followup_response = music_id.FindAlbums(album);

    /* Now our first album has full data */
    album = followup_response.Albums().at(0).next();
}
```

Objective-C:

```
GnAlbum *album = [[[response albums] allObjects] objectAtIndex:choiceOrdinal];

BOOL fullResult = [album isFullResult];

/* if we only have a partial result, we do a follow-up query to retrieve the full album */
if (!fullResult)
{
   /* do followup query to get full object. Setting the partial album as the query input. */
   GnResponseAlbums *followupResponse = [musicId findAlbumsWithGnDataObject:dataAlbum error:&error];

   /* now our first album is the desired result with full data */
   album = [[followupResponse albums] nextObject];

   // ...
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
}
```

Note that, in many cases, the data contained in a partial result is more than enough for most users and applications. For a list of values returned in a partial result, see the Data Model in the GNSDK API Reference.

Locale-Dependent Data

GNSDK provides locales as a convenient way to group locale-dependent metadata specific to a region (such as Europe) and language that should be returned from the Gracenote service. A locale is defined by a group (such as Music), a language, a region and a descriptor (indicating level of metadata detail), which are identifiers to a specific set of lists in the Gracenote Service.

There are a number of metadata fields that require having a locale loaded. For more information, see "Loading a Locale".

Accessing Enriched Content

To access enriched metadata content, such as cover art and artist images, you can purchase additional metadata entitlements and use the content URLs returned from queries in your application. Enriched content URLs can be returned in every result object (`GnAlbum`, `GnTrack`, etc).

To access enriched content:

- Purchase additional entitlements for enriched content
- Enable the query option for retrieving enriched content
- For each match object returned, iterate through its `GnContent` objects
- For each `GnContent` object, iterate through its `GnAsset` objects
- For each `GnAsset` object, get its content URL and use that for accessing the Gracenote service

Setting the Query Option for Enriched Content

To get enriched content returned from your queries, you need to enable the query option for this. You can do this using the `LookupData` method and the `kLookupDataContent` enum.

C++ example:

```
/* Enable retrieval of enriched content */
musicid.Options().LookupData(kLookupDataContent, true);
```

Objective-C example:

```
GnMusicIdStreamOptions *options = [self.gnMusicIDStream options];
[options lookupData:kLookupDataContent enable:YES error:&musicIDStreamError];
```

Processing Enriched Content

Enriched content is returned in `GnContent` objects. As indicated with these `GnContentType` enums, the following types of content can be returned:

© 2000 to present. Gracenote, Inc. All rights reserved.

- **kContentTypeImageCover**—Cover art
- **kContentTypeImageArtist**—Artist image
- **kContentTypeImageVideo**—Video image
- **kContentTypeBiography**—Artist biography
- **kContentTypeReview**—Review

Each `GnContent` object can contain one or more elements of that type as represented with a `GnAsset` object.

Note that not all enriched content can be retrieved from every metadata object. Different objects have different types of enriched content available. The following object methods are available for accessing content:

- `GnContent GnTrack::Review() const;`
- `GnContent GnAlbum::CoverArt() const;`
- `GnContent GnAlbum::Review() const;`
- `GnContent GnContributor::Image() const;`
- `GnContent GnContributor::Biography() const;`
- `gnsdk_cstr_t GnContributor::BiographyVideo() const;`—For video only

Each `GnAsset` object contains the following fields:

- **Dimension**—Asset dimension
- **Bytes**—Size of content asset in bytes
- **Size**—Pixel image size as defined with a `GnImageSize` enum, e.g., `kImageSize110` (110x110)
- **Url**—URL for retrieval of asset from Gracenote service.

Retrieving and parsing enriched content code samples:

Android Java code sample

```
        /**
         * Helpers to load and set cover art image in the application display
         */
        private void loadAndDisplayCoverArt( GnAlbum album, ImageView imageView ){
                Thread runThread = new Thread( new CoverArtLoaderRunnable( album, imageView ) );
                runThread.start();
        }
        class CoverArtLoaderRunnable implements Runnable {
                GnAlbum         album;
                ImageView       imageView;
                CoverArtLoaderRunnable( GnAlbum album, ImageView imageView){
                        this.album = album;
                        this.imageView = imageView;
                }
                @Override
                public void run() {
                        String coverArtUrl = album.content(GnContentType.
                                                kContentTypeImageCover).
                                                asset(GnImageSize.
                                                kImageSizeSmall).url();
                        Drawable coverArt = null;
                        if (coverArtUrl != null && !coverArtUrl.isEmpty()) {
                            URL url;
                            try {
                                url = new URL("http://" +
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
                                        coverArtUrl);
                                InputStream input = new
                                  BufferedInputStream(url.openStream());
                                coverArt =
                                  Drawable.createFromStream(input, "src");
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                        if (coverArt != null) {
                                setCoverArt(coverArt, imageView);
                        } else {
                                setCoverArt(getResources().getDrawable(R.drawable.no_cover_art),imageView);
                        }
                }
        }
        private void setCoverArt( Drawable coverArt, ImageView coverArtImage ){
                activity.runOnUiThread(new SetCoverArtRunnable(coverArt, coverArtImage));
        }
        class SetCoverArtRunnable implements Runnable {
                Drawable coverArt;
                ImageView coverArtImage;
                SetCoverArtRunnable( Drawable locCoverArt, ImageView locCoverArtImage) {
                        coverArt = locCoverArt;
                        coverArtImage = locCoverArtImage;
                }
                @Override
                public void run() {
                        coverArtImage.setImageDrawable(coverArt);
                }
        }
```

Objective-C code sample

```
for(GnAlbum* album in albums)
{
   /* Get CoverArt */
   GnContent *coverArtContent = [album content:kContentTypeImageCover];
   GnAsset *coverArtAsset = [coverArtContent asset:kImageSizeSmall];
   NSString *URLString = [NSString stringWithFormat:@"http://%@", [coverArtAsset url]];
   GnContent *artistImageContent = nil;//[album content:kContentTypeImageArtist];
   GnAsset *artistImageAsset = [artistImageContent asset:kImageSizeSmall];
   NSString *artistImageURLString = [NSString stringWithFormat:@"http://%@", [artistImageAsset
url]];
   GnContent *artistBiographyContent = [album content:kContentTypeBiography];
   NSString *artistBiographyURLString = [NSString stringWithFormat:@"http://%@",
[[[artistBiographyContent assets] nextObject] url]];
   GnContent *albumReviewContent = [album content:kContentTypeReview];
   NSString *albumReviewURLString = [NSString stringWithFormat:@"http://%@", [[[albumReviewContent
assets] nextObject] url]];
}
```

Windows Phone C# code sample

```
public RespAlbum(gnsdk_cppcx.GnAlbum gnAlbum, bool bTakeMachedTrack)
{
   this.Title = gnAlbum.Title.Display;
   if (false == bTakeMachedTrack)
   {
      if(0 != gnAlbum.Tracks.Count())
         this.TrackTitle = gnAlbum.Tracks.ElementAt(0).Title.Display;
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
      else
         this.TrackTitle = "";
   }
   else
   {
      if(0 != gnAlbum.TracksMatched.Count())
         this.TrackTitle = gnAlbum.TracksMatched.ElementAt(0).Title.Display;
      else
         this.TrackTitle = "";
   }
   this.ArtistName = gnAlbum.Artist.Name.Display;
   this.Genre = gnAlbum.Genre(gnsdk_cppcx.GnDataLevel.kDataLevel_1);
   this.ImageUrl = "http://" + gnAlbum.Content(gnsdk_
cppcx.GnContentType.kContentTypeImageCover).Asset(gnsdk_cppcx.GnImageSize.kImageSizeSmall).Url;
   if ("http://" == ImageUrl)
   {
      this.ImageUrl =  "/Assets/emptyImage.png";
   }
}
```

C++ code sample

```
static void
displayContentUrls(GnAlbum& album)
{
       /* content urls for album */
       content_iterator itr = album.Contents().begin();
       for (; itr != album.Contents().end(); ++itr)
       {
               GnContent content = *itr;
               switch (content.ContentType())
               {
               case kContentTypeImageCover:
                       std::cout << "cover art: " << std::endl;
                       break;
               case kContentTypeImageArtist:
                       std::cout << "artist image: " << std::endl;
                       break;
               case kContentTypeImageVideo:
                       std::cout << "video image: " << std::endl;
                       break;
               case kContentTypeBiography:
                       std::cout << "biography: " << std::endl;
                       break;
               case kContentTypeReview:
                       std::cout << "review: " << std::endl;
                       break;
               default:
                       break;
               }
               asset_iterator aItr = content.Assets().begin();
               for (; aItr != content.Assets().end(); ++aItr)
               {
                       GnAsset asset = *aItr;
                       switch (content.ContentType())
                       {
                       case kContentTypeImageCover:
                       case kContentTypeImageArtist:
                       case kContentTypeImageVideo:
                               switch (asset.Size())
                               {
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
                                default:
                                        break;
                                }
                        }
                        std::cout << "\turl: " << asset.Url() << std::endl;
                        std::cout << std::endl;
                }
        }
}
```

C# code sample

```
Under construction
```

Java code sample

```
Under construction
```

Discovery Features

Generating a Playlist

You can easily integrate the Playlist SDK into your media management application. Note that your application should already have identifiers for all its media and its own metadata database. The Playlist module allows your application to create Playlists—sets of related media—from larger collections. ***Collection Summaries***, that you create, are the basis for generating playlists. Collection Summaries contain attribute data about a given set of media. For Playlist operations—create, populate, store, delete, etc.—use the `GnPlaylistCollection` class.

To generate a Playlist:

1. Create a Collection Summary
2. Populate the Collection Summary with media objects, most likely returned from Gracenote queries
3. (Optional) Store the Collection Summary
4. (Optional) Load the stored Collection Summary into memory in preparation for Playlist results generation
5. Generate a Playlist from a Collection Summary using either the More Like This functionality or with a Playlist Description Language (PDL) statement (see the Playlist PDL Specification).
6. Access and display Playlist results.

Creating a Collection Summary

To create a Collection Summary, your application needs to instantiate a `GnPlaylistCollection` object:

C++

```
playlist::GnPlaylistCollection myCollection;
myCollection= playlist::GnPlaylistCollection("MyCollection");
```

C#

```
GnPlaylistCollection playlistCollection;
playlistCollection = GnPlaylist.CollectionCreate("sample_collection");
```

Java

© 2000 to present. Gracenote, Inc. All rights reserved.

```
GnPlaylistCollection myCollection = new GnPlaylistCollection();
```

This call creates a new, empty Collection Summary. The next step is to populate it with media items that you can use to generate Playlists.

Note: Each new Collection Summary that you create must have a unique name. Although it is possible to create more than one Collection Summary with the same name, if these Collection Summaries are then saved to local storage, one Collection will override the other. To avoid this, ensure that Collection Summary names are unique.

Populating a Collection Summary

To build a Collection Summary, your application needs to provide data for each media item you want it to contain. To add an item, and provide data, use the `GnPlaylistCollection.Add` method. This API takes a media identifier (any application-determined unique string) and an album, track or contributor match object. The match object should come from a recognition event using MusicID, MusicID-File or other GNSDK module.

C++

```
/* Create a unique identifier for every track that is added to the playlist.
   Ideally the ident allows for the identification of which track it is.
   e.g. path/filename.ext , or an id that can be externally looked up.
*/
ss.str("");
ss << index << "_" << ntrack;

/*
    Add the the Album and Track GDO for the same ident so that we can
    query the Playlist Collection with both track and album level attributes.
*/
std::string result = ss.str();
collection.Add(result.c_str(), album);     /* Add the album*/
collection.Add(result.c_str(), *itr);      /* Add the track*/
```

C#

```
/* Create a unique identifier for every track that is added to the playlist.
   Ideally, the identifier allows for the identification of which track it is.
   e.g. path/filename.ext , or an id that can be externally looked up.
*/
string uniqueIdent = countOrdinal + "_" + trackOrdinal;playlistCollection.Add(uniqueIdent, album);

/*
   Add the the Album and Track GDO for the same identifier so we can
   query the Playlist Collection with both track and album level attributes.
*/
playlistCollection.Add(uniqueIdent, album);
playlistCollection.Add(uniqueIdent, track);
```

Java

```
String uniqueIdent = "";
uniqueIdent = String.valueOf(index).concat("_").concat(String.valueOf(ntrack));

/*
 * Add the the Album and Track GDO for the same identifier so we can
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
 * query the Playlist Collection with both track and album level attributes.
 */
gnPlaylistCollection.add(uniqueIdent, gnAlbum);
gnPlaylistCollection.add(uniqueIdent, gnTrack);
```

Retrieving Playlist Attributes in Queries

When creating a MusicID or MusicID-File query to populate a playlist, you must set the following query options to ensure that the appropriate Playlist attributes are returned (depending on the type of query):

- `GnLookupData.kLookupDataSonicData`
- `GnLookupData.kLookupDataPlaylist`

C++

```
musicId.Options().LookupData(kLookupDataSonicData, true);
musicId.Options().LookupData(kLookupDataPlaylist, true);
```

How Playlist Gathers Data

When given an album match object, Playlist extracts necessary album data, traverses to the matched track, and extracts necessary track data. Playlist stores this data for the given identifier. If the album object does not contain the matched track , no track data is collected. Playlist also gathers data from both the album and track contributors as detailed below.

When given a track match object, Playlist gathers any necessary data from the track, but it is not able to gather any album-related data (such as album title). Playlist also gathers data from the track contributors as detailed below.

When given a contributor match object (or traversing into one from an album or track), Playlist gathers the necessary data from the contributor. If the contributor is a collaboration, data from both child contributors is gathered as well.

Working with Local Storage

You can store and manage Collection Summaries in local storage with the `GnPlaylistStorage`'s `Store` method, which takes a `GnPlaylistCollection` object. **Prior to doing this, your application should enable a storage solution such as SQLite.**

C++

```
/* Instantiate SQLite module to use as our database engine*/
GnStorageSqlite storageSqlite;

/* This module defaults to use the current folder when initialized,
** but we set it manually here to demonstrate the option.
 */
storageSqlite.Options().StorageLocation(".");

/* Initialize Storage for storing Playlist Collections */
playlist::GnPlaylistStorage plStorage;
plStorage.Store(myCollection);
```

C#

```
/* Initialize Storage for storing Playlist Collections */
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
playlist::GnPlaylistStorage plStorage;
gnStorage.StorageFolder = "../../../sample_db";
playlist.StoreCollection(playlistCollection);
```

Java

```
// Initialize Storage for storing Playlist Collections
GnStorageSqlite storage = new GnStorageSqlite();
GnPlaylistStorage plStorage = GnPlaylist.collectionStorage();
plStorage.store(myCollection);
```

Other `GnPlaylistStorage` methods include ones for:

- Setting a storage location specifically for playlist collections
- Removing a collection from storage
- Loading a collection from storage
- Getting the number of collections stored
- Compacting storage

Generating a Playlist Using More Like This

To streamline your Playlist implementation, you can use the `GnPlaylistCollection`'s `GenerateMoreLikeThis` method, which uses the "More Like This" algorithm to obtain results, eliminating the need to use Playlist Definition Language (PDL) statements.

You can use the `GnPlaylistMoreLikeThisOptions` class to set the following options when generating a More Like This Playlist. Please note that these options are not serialized or stored.

Option Method	Description
`MaxTracks`	Maximum number of tracks returned in a 'More Like This' playlist. Must evaluate to a number greater than 0.
`MaxPerArtist`	Maximum number of tracks per artist returned in a 'More Like This' playlist. Must evaluate to a number greater than 0.
`MaxPerAlbum`	Maximum number of results per album returned. Must evaluate to a number greater than 0.
`RandomSeed`	Randomization seed value used in calculating a More Like This playlist. Must evaluate to a number greater than 0. To re-create a playlist, use the same number - different numbers create different playlists. If "0", using a random seed is disabled.

C++

```
/* Change the possible result set to be a maximum of 30 tracks. */
collection.MoreLikeThisOptions().MaxTracks(30);

/* Change the max per artist to be 20 */
collection.MoreLikeThisOptions().MaxPerArtist(20);

/* Change the max per album to be 5 */
collection.MoreLikeThisOptions().MaxPerAlbum(5);
```

© 2000 to present. Gracenote, Inc. All rights reserved.

To generate a More Like This playlist, call the `GnPlaylistCollection`'s `GenerateMoreLikeThis` method with a user object handle and a Gracenote data object, such as the object of the track that is currently playing.

C++

```
/* Generating more like this Playlist */
/* Create seed data to generate more like this playlist*/
/*
 * A seed gdo can be any recognized media gdo.
 * In this example, we are using a gdo from a track in the playlist collection summary,
 * randomly selecting the 5th element
 */
playlist::GnPlaylistIdentifier ident      = collection.MediaIdentifiers().at(4).next();
playlist::GnPlaylistMetadata   seed_album = collection.Metadata(user, ident);
playlist::GnPlaylistResult resultMoreLikeThis = collection.GenerateMoreLikeThis(user, seed_album,
topColl));
```

C#

```
/*
 * A seed gdo can be any recognized media gdo.
 * In this example we are using the a gdo from a random track in the playlist collection summary
 */
GnPlaylistIdentifier identifier = playlistCollection.MediaIdentifiers.at(3).Current;
GnPlaylistMetadata data = playlistCollection.Metadata(user, identifier);
playlistCollection.MoreLikeThisOptionSet
(GnPlaylistCollection.GenerateMoreLikeThisOption.kMoreLikeThisMaxPerAlbum, 5);
GnPlaylistResult playlistResult = playlistCollection.GenerateMoreLikeThis(user, data);
```

Java

```
GnPlaylistMetadata getSeedData( GnUser gnUser, GnPlaylistCollection collection ) throws GnException
{
   // Create seed data to generate more like this playlist
   // A seed gdo can be any recognized media gdo.
   // In this example we are using the a gdo from a track in the playlist collection summary
   // In this case , randomly selecting the 5th element
   GnPlaylistIdentifier ident      = collection.mediaIdentifiers().at(4).next();
   GnPlaylistMetadata   seedAlbum  = collection.metadata( gnUser, ident );
   return seedAlbum;
}

//...

// Set options
collection.options().moreLikeThis( GnPlaylistMoreLikeThisOption.kMoreLikeThisMaxTracks, 30 );
collection.options().moreLikeThis( GnPlaylistMoreLikeThisOption.kMoreLikeThisMaxPerArtist, 10 );
collection.options().moreLikeThis( GnPlaylistMoreLikeThisOption.kMoreLikeThisMaxPerAlbum, 5 );

GnPlaylistResult resultCustomMoreLikeThis = collection.generateMoreLikeThis( gnUser, getSeedData(
gnUser, collection ) );
```

Generating a Playlist Using PDL (Playlist Description Language)

The GNSDK Playlist Definition Language (PDL) is a query syntax, similar to Structured Query Language (SQL), that enables flexible custom playlist generation using human-readable text strings. PDL allows developers to dynamically create custom playlists. By storing individual PDL statements, applications can create and manage multiple preset and user playlists for later use.

© 2000 to present. Gracenote, Inc. All rights reserved.

PDL statements express the playlist definitions an application uses to determine what items are included in resulting playlists. PDL supports logic operators, operator precedence, and in-line arithmetic. PDL is based on Structured Query Language (SQL). This section assumes you understand SQL and can write SQL statements.

Note: Before using PDL statements, carefully consider if the provided More Like This functionality meets your design requirements. More Like This functionality eliminates the need to create and validate PDL statements.

To generate a playlist using PDL, you can use the `GnPlaylistCollections`'s `GeneratePlaylist` method which takes a PDL statement as a string and returns `GnPlaylistResult` objects.

To understand how to create a PDL statement, see the *Playlist PDL Specification* article.

Accessing Playlist Results

When you generate a Playlist from a Collection Summary, using either More Like This or executing a PDL statement, results are returned in a `GnPlaylistResult` object.

C++

```
/*-----------------------------------------------------------------------------
 *  display_playlist_results
 */
void display_playlist_results(GnUser& user, playlist::GnPlaylistCollection& collection,
playlist::GnPlaylistResult& result)
{
   /* Generated playlist count */
   int resultCount = result.Identifiers().count();

   printf("Generated Playlist: %d\n", resultCount);
   playlist::result_iterator itr = result.Identifiers().begin();

   for (; itr != result.Identifiers().end(); ++itr)
   {
      playlist::GnPlaylistMetadata data = collection.Metadata(user, *itr);

      printf("Ident '%s' from Collection '%s':\n"
             "\tGN_AlbumName  : %s\n"
             "\tGN_ArtistName : %s\n"
             "\tGN_Era        : %s\n"
             "\tGN_Genre      : %s\n"
             "\tGN_Origin     : %s\n"
             "\tGN_Mood       : %s\n"
             "\tGN_Tempo      : %s\n",
             (*itr).MediaIdentifier(),
             (*itr).CollectionName(),
              data.AlbumName(),
              data.ArtistName(),
              data.Era(),
              data.Genre(),
              data.Origin(),
              data.Mood(),
              data.Tempo()
       );
   }
}
```

© 2000 to present. Gracenote, Inc. All rights reserved.

C#

```
private static void
EnumeratePlaylistResults(GnUser user, GnPlaylistCollection playlistCollection, GnPlaylistResult
playlistResult)
{
    GnPlaylistMetadata gdoAttr = null;
    string ident = null;
    string collectionName = null;
    uint countOrdinal = 0;
    uint resultsCount = 0;
    GnPlaylistCollection tempCollection = null;

    resultsCount = playlistResult.Identifiers.count();

    Console.WriteLine("Generated Playlist: " + resultsCount);

    GnPlaylistResultIdentEnumerable playlistResultIdentEnumerable = playlistResult.Identifiers;
    foreach (GnPlaylistIdentifier playlistIdentifier in playlistResultIdentEnumerable)
    {
        collectionName = playlistIdentifier.CollectionName;
        ident = playlistIdentifier.MediaIdentifier;

        Console.Write("    " + ++countOrdinal + ": " + ident + " Collection Name:" +
collectionName);

        /* The following illustrates how to get a collection handle
           from the collection name string in the results enum function call.
           It ensures that Joined collections as well as non joined collections will work with
minimal overhead.
        */
        tempCollection = playlistCollection.JoinSearchByName(collectionName);

        gdoAttr = tempCollection.Metadata(user, playlistIdentifier);

        PlaylistGetAttributeValue(gdoAttr);

    }
}

private static void
PlaylistGetAttributeValue(GnPlaylistMetadata gdoAttr)
{
    /* Album name */
    if (gdoAttr.AlbumName != "")
        Console.WriteLine("\n\t\tGN_AlbumName:" + gdoAttr.AlbumName);

    /* Artist name */
    if (gdoAttr.ArtistName != "")
        Console.WriteLine("\t\tGN_ArtistName:" + gdoAttr.ArtistName);

    /* Artist Type */
    if (gdoAttr.ArtistType != "")
        Console.WriteLine("\t\tGN_ArtistType:" + gdoAttr.ArtistType);

    /*Artist Era */
    if (gdoAttr.Era != "")
        Console.WriteLine("\t\tGN_Era:" + gdoAttr.Era);

    /*Artist Origin */
    if (gdoAttr.Origin != "")
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
        Console.WriteLine("\t\tGN_Origin:" + gdoAttr.Origin);

    /* Mood */
    if (gdoAttr.Mood != "")
        Console.WriteLine("\t\tGN_Mood:" + gdoAttr.Mood);

    /*Tempo*/
    if (gdoAttr.Tempo != "")
        Console.WriteLine("\t\tGN_Tempo:" + gdoAttr.Tempo);
}
```

Java

```
private static void enumeratePlaylistResults(
    GnUser user,
    GnPlaylistCollection playlistCollection,
    GnPlaylistResult playlistResult
) throws GnException {
    int countOrdinal = 0;
    GnPlaylistMetadata data = null;
    GnPlaylistCollection tempCollection = null;
    String collectionName = null;
    System.out.println("Generated Playlist: " + playlistResult.identifiers().count());

    GnPlaylistResultIdentIterator gnPlaylistResultIdentIterator = playlistResult.identifiers().begin
();

    /* Iterate through results */
       while (gnPlaylistResultIdentIterator.hasNext()) {
        GnPlaylistIdentifier playlistIdentifier = gnPlaylistResultIdentIterator.next();

        collectionName = playlistIdentifier.collectionName();

        System.out.println("    "+ ++countOrdinal +": "+playlistIdentifier.mediaIdentifier()+
                           " Collection Name:"+playlistIdentifier.collectionName());

        /* The following illustrates how to get a collection handle
           from the collection name string in the results enum function call.
           It ensures that Joined collections as well as non joined collections will work with
minimal overhead.
        */
        tempCollection = playlistCollection.joinSearchByName(collectionName);

        data = tempCollection.metadata(user, playlistIdentifier);

        playlistGetAttributeValue(data);
    }
}

private static void playlistGetAttributeValue(GnPlaylistMetadata data) {

    /* Album name */
    if (data.albumName() != "" && data.albumName() != null)
        System.out.println("\t\tGN_AlbumName:" + data.albumName());

    /* Artist name */
    if (data.artistName() != "" && data.artistName() != null)
        System.out.println("\t\tGN_ArtistName:" + data.artistName());

    /* Artist Type */
    if (data.artistType() != "" && data.artistType() != null)
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
        System.out.println("\t\tGN_ArtistType:" + data.artistType());

    /*Artist Era */
    if (data.era() != "" && data.era() != null)
        System.out.println("\t\tGN_Era:" + data.era());

    /*Artist Origin */
    if (data.origin() != "" && data.origin() != null)
        System.out.println("\t\tGN_Origin:" + data.origin());

    /* Mood */
    if (data.mood() != "" && data.mood() != null)
        System.out.println("\t\tGN_Mood:" + data.mood());

    /*Tempo*/
    if (data.tempo() != "" && data.tempo() != null)
        System.out.println("\t\tGN_Tempo:" + data.tempo());
}
```

Working with Multiple Collection Summaries

Creating a playlist across multiple collections can be accomplished by using joins. Joins allow you to combine multiple collection summaries at run-time, so that they can be treated as one collection by the playlist generation functions. Joined collections can be used to generate More Like This and PDL-based playlists.

For example, if your application has created a playlist based on one device (collection 1), and another device is plugged into the system (collection 2), you might want to create a playlist based on both of these collections. This can be accomplished using one of the `GnPlaylistCollection` join methods.

Joins are run-time constructs for playlist generation that support seamless identifier enumeration across all contained collections. They do not directly support the addition or removal of data objects, synchronization, or serialization across all collections in a join. To perform any of these operations, you can use the join management functions to access the individual collections and operate on them separately.

To remove a collection from a join, call the `GnPlaylistCollection`'s `JoinRemove` method.

Join Performance and Best Practices

Creating a join is very efficient and has minimal CPU and memory requirements. When collections are joined, GNSDK internally sets references between them, rather than recreating them. Creating, deleting, and recreating joined collections when needed can be an effective and high-performing way to manage collections.

The original handles for the individual collections remain functional, and you can continue to operate on them in tandem with the joined collection, if needed. If you release an original handle for a collection that has been entered into a joined collection, the joined collections will continue to be functional as long as the collection handle representing the join remains valid.

A good practice for managing the joining of collections is to create a new collection handle that represents the join, and then join all existing collections into this handle. This helps remove ambiguity as to which original collection is the parent collection representing the join.

© 2000 to present. Gracenote, Inc. All rights reserved.

Synchronizing Collection Summaries

Collection summaries must be refreshed whenever items in the user's media collection are modified. For example, if you've created a collection summary based on the media on a particular device, and the media on that device changes, your application must synchronize the Collection Summary.

1. Adding all existing (current and new) unique identifiers, using the `GnPlaylistCollection`'s `SyncProcessAdd` method.
2. Calling the `GnPlaylistCollection`'s `SyncProcessExecute` method to process the current and new identifiers and using the callback function to add or remove identifiers to or from the Collection Summary.

Iterating the Physical Media

The first step in synchronizing is to iterate through the physical media, calling the `GnPlaylistCollection`'s `SyncProcessAdd` method for each media item. For each media item, pass the unique identifier associated with the item to the method. The unique identifiers used must match the identifiers that were used to create the Collection Summary initially.

Processing the Collection

After preparing a Collection Summary for synchronization using the `GnPlaylistCollection`'s `SyncProcessAdd` method, call the `GnPlaylistCollection`'s `SyncProcessExecute` method to synchronize the Collection Summary's data. During processing, the callback function will be called for each difference between the physical media and the collection summary. This means the callback function will be called once for each new media item, and once for each media item that has been deleted from the collection. The callback function should add new and delete old identifiers from the Collection Summary.

Playlist PDL Specification

The GNSDK for Mobile Playlist Definition Language (PDL) is a query syntax, similar to Structured Query Language (SQL), that enables flexible custom playlist generation using human-readable text strings. PDL allows developers to dynamically create custom playlists. By storing individual PDL statements, applications can create and manage multiple preset and user playlists for later use.

PDL statements express the playlist definitions an application uses to determine what items are included in resulting playlists. PDL supports logic operators, operator precedence and in-line arithmetic. PDL is based on Search Query Language (SQL). This section assumes you understand SQL and can write SQL statements.

NOTE: Before implementing PDL statement functionality for your application, carefully consider if the provided More Like This function, gnsdk_playlist_generate_morelikethis( ) meets your design requirements. Using the More Like This function eliminates the need to create and validate PDL statements.

PDL Syntax

This topic discusses PDL keywords, operators, literals, attributes, and functions.

© 2000 to present. Gracenote, Inc. All rights reserved.

Note: Not all keywords support all operators. Use gnsdk_playlist_statement_validate() to check a PDL Statement, which generates an error for invalid operator usage.

Keywords

PDL supports these keywords:

Keyword	Description	Required or Optional	PDL Statement Order
GENERATE PLAYLIST	All PDL statements must begin with either GENERATE PLAYLIST or its abbreviation, GENPL	Required	1
WHERE	Specifies the attributes and threshold criteria used to generate the playlist. If a PDL statement does not include the WHERE keyword, Playlist operates on the entire collection.	Optional	2
ORDER	Specifies the criteria used to order the results' display. If a PDL statement does not include the ORDER keyword, Playlist returns results in random order. Example: Display results in based on a calculated similarity value; tracks having greater similarity values to input criteria display higher in the results. The expression format is: <identifier> <operator> <identifier>	Optional	3
LIMIT	Specifies criteria used to restrict the number of returned results. Also uses the keywords RESULT and PER. Example: Limiting the number of tracks displayed in a playlist to 30 results with a maximum of two tracks per artist. The expression format is: <identifier> <operator> <identifier>	Optional	4

© 2000 to present. Gracenote, Inc. All rights reserved.

Keyword	Description	Required or Optional	PDL Statement Order
SEED	Specifies input data criteria from one or more idents. Typically, a Seed is the This in a More Like This playlist request. Example: Using a Seed of Norah Jones' track Don't Know Why to generate a playlist of female artists of a similar genre. The expression format is: <identifier> <operator> <identifier>	Optional	NA

Example: Keyword Syntax

This PDL statement example shows the syntax for each keyword. In addition to <att_imp>, <expr>, and <score> discussed above, this example shows:

- <math_op> is one of the valid PDL mathematical operators.
- <number> is positive value.
- <attr_name> is a valid attribute, either a Gracenote-delivered attribute or an implementation-specific attribute.

```
GENERATE PLAYLIST
 WHERE <att_imp> [<math_op> <score>][ AND|OR <att_imp>]
ORDER <expr>[ <math_op> <expr>]
 LIMIT [number RESULT | PER <attr_name>][,number [ RESULT | PER <attr_name>]]
```

Operators

PDL supports these operators:

Operator Type	Available Operators
Comparison	>, >=, <, <=, ==, !=, LIKE LIKE is for fuzzy matching, best used with strings; see PDL Examples
Logical	AND, OR
Mathematical	+, -, *, /

Literals

PDL supports the use of single (') and double (") quotes, as shown:

- Single quotes: 'value one'
- Double quotes: "value two"

© 2000 to present. Gracenote, Inc. All rights reserved.

- Single quotes surrounded by double quotes: "'value three'"

> You must enclose a literal in quotes or Playlist evaluates it as an attribute.

Attributes

Most attributes require a Gracenote-defined numeric identifier value or GDO Seed.

- Identifier: Gracenote-defined numeric identifier value is typically a 5-digit value; for example, the genre identifier 24045 for Rock. These identifiers are maintained in lists in Gracenote Service; download the lists using GNSDK for Mobile Manager's Lists and List Types APIs
- GDO Seed: Use GNSDK Manager's GDO APIs to access XML strings for Seed input.

The following table shows the supported system attributes and their respective required input. The first four attributes are the GOET attributes.

> The delivered attributes have the prefix GN_ to denote they are Gracenote standard attributes. You can extend the attribute functionality in your application by implementing custom attributes; however, do not use the prefix GN_.

Name	Attribute	Required Input
Genre Origin Era Artist Type Mood Tempo	GN_Genre GN_Origin GN_Era GN_ArtistType GN_Mood GN_Tempo	Gracenote-defined numeric identifier value GDO Seed XML string
Artist Name Album Name	GN_ArtistName GN_AlbumName	Text string

Functions

PDL supports these functions:

- RAND(max value)
- RAND(max value, seed)

PDL Statements

This topic discusses PDL statements and their components.

© 2000 to present. Gracenote, Inc. All rights reserved.

Attribute Implementation <att_imp>

A PDL statement is comprised of one or more attribute implementations that contain attributes, operators, and literals. The general statement format is:

"GENERATE PLAYLIST WHERE <attribute> <operator> <criteria>"

You can write attribute implementations in any order, as shown:

GN_ArtistName == "ACDC"

or

"ACDC" == GN_ArtistName

WHERE and ORDER statements can evaluate to a score; for example:

"GENERATE PLAYLIST WHERE LIKE SEED > 500"

WHERE statements that evaluate to a non-zero score determine what idents are in the playlist results. ORDER statements that evaluate to a non-zero score determine how idents display in the playlist results.

Expression <expr>

An expression performs a mathematical operation on the score evaluated from a attribute implementation.

[<number> <math_op>] <att_imp>

For example:

3 * (GN_Era LIKE SEED)

Score <score>

Scores can range between -1000 and 1000.

For boolean evaluation, True equals 1000 and False equals 0.

Note: For more complex statement scoring, concatenate attribute implementations and add weights to a PDL statement.

Example: PDL Statements

The following PDL example generates results that have a genre similar to and on the same level as the seed input. For example, if the Seed genre is a Level 2: Classic R&B/Soul, the matching results will include similar Level 2 genres, such as Neo-Soul.

```
"GENERATE PLAYLIST WHERE GN_Genre LIKE SEED"
```

This PDL example generates results that span a 20-year period. Matching results will have an era value from the years 1980 to 2000.

```
"GENERATE PLAYLIST WHERE GN_Era >= 1980 AND GN_Era < 2000"
```

© 2000 to present. Gracenote, Inc. All rights reserved.

This PDL example performs fuzzy matching with Playlist, by using the term LIKE and enclosing a string value in single (') or double (") quotes (or both, if needed). It generates results where the artist name may be a variation of the term ACDC, such as:

- ACDC
- AC/DC
- AC*DC

```
"GENERATE PLAYLIST WHERE (GN_ArtistName LIKE 'ACDC')"
```

The following PDL example generates results where:

- The tempo value must be less than 120 BPM.
- The ordering displays in descending order value, from greatest to least (119, 118, 117, and so on).
- The genre is similar to the Seed input.

```
"GENERATE PLAYLIST WHERE GN_Tempo > 120 ORDER GN_Genre LIKE SEED"
```

Implementing Moodgrid

The MoodGrid library allows applications to generate playlists and user interfaces based on Gracenote Mood descriptors. MoodGrid provides Mood descriptors to the application in a two-dimensional grid that represents varying degrees of moods across each axis. One axis represents energy (calm to energetic) and the other axis represents valence (dark to positive). When the user selects a mood from the grid, the application can provide a playlist of music that corresponds to the selected mood. Additional filtering support is provided for genre, origin, and era music attributes.

For more information on Moodgrid, see the *Moodgrid Overview*

The Moodgrid APIs:

- Encapsulate Gracenote's Mood Editorial Content (mood layout and ids).
- Simplify access to MoodGrid results through x,y coordinates.
- Allow for multiple local and online data sources through MoodGrid Providers.
- Enable pre-filtering of results using genre, origin, and era attributes.
- Support 5x5 or 10x10 MoodGrids.
- Provide the ability to go from a cell of a 5x5 MoodGrid to any of its expanded four Moods in a 10x10 grid.

Implementing MoodGrid in an application involves the following steps:

1. Allocating a `GnMoodgrid` class.
2. Enumerating the data sources using MoodGrid Providers
3. Creating and populating a MoodGrid Presentation
4. Filtering the results, if needed

© 2000 to present. Gracenote, Inc. All rights reserved.

Prerequisites

Using the MoodGrid APIs requires the following modules:

- GNSDK Manager
- SQLite (for local caching)
- MusicID
- Playlist
- MoodGrid

If you are using MusicID to recognize music, you must enable Playlist data in your query. You must be entitled to use Playlist—if you are not, you will not get an error, but MoodGrid will return no results. Please contact your Gracenote Global Service & Support representative for more information.

Enumerating Data Sources using MoodGrid Providers

GNSDK automatically registers all local and online data sources available to MoodGrid. For example, if you create a playlist collection using the Playlist API, GNSDK automatically registers that playlist as a data source available to MoodGrid. These data sources are referred to as *Providers*. MoodGrid is designed to work with multiple providers. You can iterate through the list of available Providers using the `GnMoodgrid` class' providers method. For example, the following call returns a handle to the first Provider on the list (at index 0):

C++ code sample

```
moodgrid::GnMoodgrid myMoodgrid;
moodgrid::GnMoodgridProvider myProvider = *(myMoodgrid.Providers().at(0));
```

C# code sample

```
GnMoodgrid moodgrid = new GnMoodgrid();
GnMoodgridProvider provider = moodgrid.Providers.at(0).next();
```

Java code sample

```
GnMoodgrid myMoodGrid = new GnMoodgrid();
GnMoodgridProvider myProvider = myMoodGrid.providers().at(0).next();
```

You can use the `GnMoodGridProvider` object to retrieve the following information

- Name
- Type
- Requires network

Creating and Populating a MoodGrid Presentation

Once you have a `GnMoodgrid` object, you can create and populate a MoodGrid Presentation with Mood data. A Presentation is a `GnMoodgridPresentation` object that represents the MoodGrid, containing the mood name and playlist information associated with each grid cell.

To create a MoodGrid Presentation, use the `GnMoodgrid` class' "create presentation" method, passing in the user handle and the MoodGrid type. The type can be one of the enumerated values in `GnMoodgridPresentationType`: either a 5x5 or 10x10 grid. The method returns a `GnMoodgridPresentation` object:

© 2000 to present. Gracenote, Inc. All rights reserved.

C++ code sample

```
moodgrid::GnMoodgridPresentation myPresentation =  myMoodgrid.CreatePresentation(user,
GnMoodgridPresentationType.kMoodgridPresentationType5x5);
```

C# code sample

```
presentation = moodGrid.CreatePresentation(user,
GnMoodgridPresentationType.kMoodgridPresentationType10x10);
```

Java code sample

```
GnMoodgridPresentation myPresentation = myMoodGrid.createPresentation(user,
GnMoodgridPresentationType.kMoodgridPresentationType5x5);
```

Iterating Through a MoodGrid Presentation

Each cell of the Presentation is populated with a mood name an associated playlist. You can iterate through the Presentation to retrieve this information from each cell:

C++ code sample

```
/* Create a moodgrid presentation for the specified type */
moodgrid::GnMoodgridPresentation myPresentation =  myMoodgrid.CreatePresentation(user, type);

moodgrid::GnMoodgridPresentation::data_iterator itr = myPresentation.Moods().begin();

for (; itr != myPresentation.Moods().end(); ++itr) {

    /* Find the recommendation for the mood */
    moodgrid::GnMoodgridResult result = myPresentation.FindRecommendations(myProvider, *itr);

    printf("\n\n\tX:%d  Y:%d\tMood Name: %s\tMood ID: %s\tCount: %d\n", itr->X, itr->Y,
myPresentation.MoodName(*itr), myPresentation.MoodId(*itr), result.Count());

    moodgrid::GnMoodgridResult::iterator result_itr = result.Identifiers().begin();

    /* Iterate the results for the identifiers */
    for (; result_itr != result.Identifiers().end(); ++result_itr) {
        printf("\n\n\tX:%d  Y:%d", itr->X, itr->Y);

        printf("\nident:\t%s\n", result_itr->MediaIdentifier());
        printf("group:\t%s\n", result_itr->Group());

        playlist::GnPlaylistMetadata   data = collection.Metadata(user, result_itr->MediaIdentifier
(), result_itr->Group());

        printf("Album:\t%s\n", data.AlbumName());
        printf("Mood :\t%s\n", data.Mood());
    }
}
```

C# code sample

```
/* Create a moodgrid presentation for the specified type */
presentation = moodGrid.CreatePresentation(user, gnMoodgridPresentationType);

/* Query the presentation type for its dimensions */
GnMoodgridDataPoint dataPoint = moodGrid.Dimensions(gnMoodgridPresentationType);
Console.WriteLine("\n PRINTING MOODGRID " + dataPoint.X + " x " + dataPoint.Y + " GRID ");
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
/* Enumerate through the moodgrid getting individual data and results */
GnMoodgridPresentationDataEnumerable moodgridPresentationDataEnumerable = presentation.Moods;


foreach (GnMoodgridDataPoint position in moodgridPresentationDataEnumerable)
{
    uint x = position.X;
    uint y = position.Y;

    /* Get the name for the grid coordinates in the language defined by Locale */
    string name = presentation.MoodName(position);

    /* Get the mood id */
    string id = presentation.MoodId(position);

    /* Find the recommendation for the mood */
    GnMoodgridResult moodgridResult = presentation.FindRecommendations(provider, position);

    /* Count the number of results */
    count = moodgridResult.Count();
    Console.WriteLine("\n\n\tX:" + x + "  Y:" + y + " name: " + name + " count: " + count + " ");

    /* Iterate the results for the idents */
    GnMoodgridResultEnumerable identifiers = moodgridResult.Identifiers;
    foreach (GnMoodgridIdentifier identifier in identifiers)
    {
        string ident = identifier.MediaIdentifier;
        string group = identifier.Group;
        Console.WriteLine("\n\tX:" + x + " Y:" + y + " \nident:\t" + ident + "  \ngroup:\t" + group
);
    }
}
```

Java code sample

```
/* Create a moodgrid presentation for the specified type */
GnMoodgridPresentation myPresentation = myMoodGrid.createPresentation(user, type);

/* Query the presentation type for its dimensions */
GnMoodgridDataPoint dataPoint = myMoodGrid.dimensions(type);
System.out.println("\n PRINTING MOODGRID " + dataPoint.getX() + " x " + dataPoint.getY() + " GRID
");

GnMoodgridPresentationDataIterator itr = myPresentation.moods().begin();

while(itr.hasNext()) {
    GnMoodgridDataPoint position = itr.next();

    /* Find the recommendation for the mood */
    GnMoodgridResult moodgridResult = myPresentation.findRecommendations(myProvider, position);

    System.out.println("\n\n\tX:" + position.getX()
        + "  Y:" + position.getY()
        + " name: "+ myPresentation.moodName(position)
        + " count: " + moodgridResult.count()
        + " ");

    GnMoodgridResultIterator resultItr = moodgridResult.identifiers().begin();

    while(resultItr.hasNext()) {
        GnMoodgridIdentifier resultIdentfier = resultItr.next();
```

© 2000 to present. Gracenote, Inc. All rights reserved.

```
        System.out.println("\n\tX:" + position.getX() + " Y:" + position.getY()+" ");
        System.out.println("ident:\t" + resultIdentfier.mediaIdentifier()+"  ");
        System.out.println("group:\t" + resultIdentfier.group());

    }
}
```

Filtering MoodGrid Results

You can use genre, origin, and era to filter MoodGrid results. If you apply a filter, the results that are returned are pre-filtered, reducing the amount of data transmitted. For example, the following call sets a filter to limit results to tracks that fall within the Rock genre.

C++ code sample

```
Under construction
```

C# code sample

```
Under construction
```

Java code sample

```
Under construction
```

Implementing Rhythm

You can use the Rhythm API to:

- Create radio stations with a *seed* value.
- Generate a playlist with or without having to create a radio station.
- Adjust a radio station's playlist with feedback events and tuning parameters

Using Rhythm in your application involves doing some or all of the following:

1. Creating a Rhythm query with initial seed value.
2. Set Rhythm query options to fine-tune the results you want to see returned.
3. Generating a playlist, either as recommendations or when creating a radio station.
4. Retrieving the playlist.
5. For radio stations:
   - Sending feedback events (track/artist like/dislike/skip etc.) which may alter the station's playlist queue.
   - Retrieving playlists altered from feedback events
   - Saving a radio station for later retrieval.

Creating a Rhythm Query

Before using Rhythm, follow the usual steps to initialize GNSDK for Mobile. The following GNSDK for Mobile modules must be initialized:

© 2000 to present. Gracenote, Inc. All rights reserved.

- GNSDK Manager
- MusicID (optional to retrieve GnDataObjects)
- Playlist (optional to use generated playlists)
- Rhythm

For more information on initializing the GNSDK for Mobile modules, see "Initializing an Application."

To create a Rhythm query:

1. Instantiate a `GnRhythmQuery` object with your User object.
2. Call the `GnRhythmQuery`'s `AddSeed` method with a `GnDataObject` seed that you can get with a MusicID query.

Code Samples

C++

```
// Instantiate the query with a Gracenote user object
GnRhythmQuery rhythmQuery(user);

// Assume we start with a serialized data object (from MusicID)
// to create our seed.
GnDataObject seed = GnDataObject::Deserialize(serializedDataObject);
rhythmQuery.AddSeed(seed);
```

Java

```
// Instantiate the query with a Gracenote User object
GnRhythmQuery rhythmQuery   = new GnRhythmQuery(user);
// Assume we start with a serialized data object (from MusicID)
// to create our seed.
GnDataObject seed = GnDataObject.deserialize(serializedDataObject);
rhythmQuery.addSeed(seed);
```

Setting Rhythm Query Options

You can use the following `GnRhythmQueryOptions` methods to set options for a Rhythm Query.

- **`FocusPopularity`**—Set wth a number indicating how popular tracks should be. Range is 0-1000. Default is 1000 (very popular).
- **`FocusSimilarity`**—Set wth a number indicating how similar tracks should be. Range is 0-1000. Default is 1000 (very similar).
- **`LookupData`**—Request additional content in your results: classical, sonic, external IDs (3rd-party) identifiers, global IDs, credits, etc. See the `GnLookupData` enums for a complete list.
- **`RecommendationMaxTracksPerArtist`**—Specify a maximum number of tracks per artist for recommended playlist results. Range is 1-1000.
- **`RecommendationRotationRadio`**—Specifying `true` will cause results to be sequenced in a radio-like fashion. Choosing `false` will return normal recommendation results without sequencing. Default is `false`.
- **`ReturnCount`**—Specify how many tracks can be returned. The range is 1-25. Default is 5.

© 2000 to present. Gracenote, Inc. All rights reserved.

- **`StationDMCA`**—DMCA stands for Digital Millenium Copyright Act. When creating a radio station, you have the option to enable DMCA rules, which reduces the repetition of songs and albums in conformance with DMCA guidelines.

Generating Recommendations

Once you have a seeded query, you can create a recommendations playlist, which uses less overhead than also creating a radio station

To generate a Rhythm query recommendations playlist:

1. Call your `GnRhythmQuery` object"s `GenerateRecommendations` method.

   C++

   ```
   GnResponseAlbums recommendations = rhythmQuery.GenerateRecommendations();
   ```

   Java

   ```
   GnResponseAlbums recommendations = rhythmQuery.generateRecommendations();
   ```

2. Iterate through the generated `GnResponseAlbums` object.

Once you have a playlist, you can iterate through it to get album and track information.

Creating a Radio Station and Playlist

Creating a radio station and a playlist allows you to modify the playlist through feedback or tuning.

To create a radio station and playlist:

1. Instantiate a `GnRhythmStation` object, using your `GnRhythmQuery` object:

   C++

   ```
   GnRhythmStation rhythmStation(rhythmQuery);
   ```

   Java

   ```
   GnRhythmStation rhythmStation = new GnRhythmStation(rhythmQuery);
   ```

   The Rhythm station object is how you provide feedback and tuning information.

2. To generate a radio station playlist, use the `GnRhythmStation` `GeneratePlaylist` method:

   C++

   ```
   GnResponseAlbums playlist = rhythmStation.GeneratePlaylist();
   ```

   Java

   ```
   GnResponseAlbums playlist = rhythmStation.generatePlaylist();
   ```

© 2000 to present. Gracenote, Inc. All rights reserved.

Providing Feedback

Once you have created a station, you cannot generate a new playlist for it with a different seed. To use a different seed you would have to create a new station. However, you can use feedback events to modify an existing station's playlist. The following `GnRhythmEvent` enums indicate the supported feedback events and how they affect a radio station's playlist:

- **`kRhythmEventTrackPlayed`**— Track marked as played. Advances the play queue (drops track being played and adds additional track to end of queue).
- **`kRhythmEventTrackSkipped`**— Track marked as skipped. Advances the queue by one.
- **`kRhythmEventTrackLike`**— Track marked as liked. Does not change the playlist queue.
- **`kRhythmEventTrackDislike`**— Track marked as disliked. Modifies the playlist queue but does not drop the disliked track. To do that, combine this with a track skipped event.
- **`kRhythmEventArtistLike`**— Artist marked as liked. Does not change the playlist queue.
- **`kRhythmEventArtistDislike`**— Artist marked as disliked. Modifies the playlist queue.

For a more detailed explanation on how feedback events modify a playlist, reference the Rhythm API documentatioin on the Gracenote developer portal:
https://developer.gracenote.com/sites/default/files/web/rhythm/index.htm

See the *How Feedback Events Affect the Queue* article.

To provide feedback about a Station's playlist use the Station object's `Event` method. This method takes a `GnRhythmEvent` enum and a `GnDataObject`. You set the event, and either the track or artist for the event. For example, the following two functions send event information about a track or an artist to a station.

Code samples

C++

```
GnArtist artist = playlist.Albums().at(1)->Artist();
rhythmStation.Event(kRhythmEventArtistDislike, artist);

GnTrack track = playlist.Albums().at(1)->TrackMatched();
rhythmStation.Event(kRhythmEventTrackLike, track);
```

Java

```
GnArtist artist = playlist.albums().at(1).next().artist();
rhythmStation.event(kRhythmEventArtistDislike, artist);

GnTrack track = playlist.albums().at(1).next().trackMatched();
rhythmStation.event(kRhythmEventTrackLike, track);
```

Tuning a Radio Station

You can set options to adjust a radio station and its playlist after initial creation. Use the Rhythm station object's `Options()` methods to change values. These options are the same as the Rhythm query options.

For example, the following C++ code turns DMCA support on:

© 2000 to present. Gracenote, Inc. All rights reserved.

```
rhythmStation.Options().StationDMCA(true);
```

Saving and Retrieving Radio Stations

Rhythm saves all created radio stations within the Gracenote service. Stations can be recalled through the station ID, which your application can store locally. To retrieve the station ID for storage, use the Station object's `StationId()` method.

Once you have the station ID, an application can save it to persistent storage and recall it at any later date. Station IDs are permanently valid. To retrieve a station create a `GnRhythmStation` object with the Station ID and associated user. This retrieval can take place during your app's current running or any future running.

For example, the following C++ code saves and retrieves a Station:

```
// Get station ID

gnsdk_cstr_t stationId = rhythmStation.StationId();

// Save the ID for later retrieval...

// Create a station based on the saved station ID
GnRhythmStation rhythmStation(stationId, user);
```

Best Practices and Design Requirements

Image Best Practices

Gracenote images – in the form of cover art, artist images and more – are integral features in many online music services, as well as home, automotive and mobile entertainment devices. Gracenote maintains a comprehensive database of images in dimensions to accommodate all popular applications, including a growing catalog of high-resolution (HD) images.

Gracenote carefully curates images to ensure application and device developers are provided with consistently formatted, high quality images – helping streamline integration and optimize the end-user experience. This topic describes concepts and guidelines for Gracenote images including changes to and support for existing image specifications.

Image Resizing Guidelines

Gracenote images are designed to fi

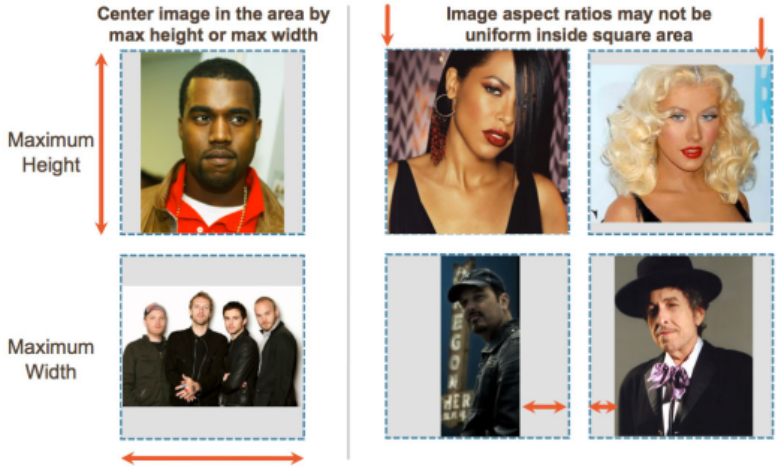
t within squares as defined by the available image dimensions. This allows developers to present images in a fixed area within application or device user interfaces. Gracenote recommends applications center images horizontally and vertically within the predefined square dimensions, and that the square be transparent such that the background shows through. This results in a consistent presentation despite variation in the image dimensions. To ensure optimum image quality for end-users, Gracenote recommends that applications use Gracenote images in their provided pixel dimensions without stretching or resizing.

Gracenote resizes images based on the following guidelines:

© 2000 to present. Gracenote, Inc. All rights reserved.

- **Fit-to-square**: images will be proportionally resized to ensure their largest dimension (if not square) will fit within the limits of the next lowest available image size.
- **Proportional resizing**: images will always be proportionally resized, never stretched.
- **Always downscale**: smaller images will always be generated using larger images to ensure the highest possible image quality

Following these guidelines, all resized images will remain as rectangles retaining the same proportions as the original source images. Resized images will fit into squares defined by the available dimensions, but are not themselves necessarily square images.



For Tribune Media Services (TMS) video images only, Gracenote will upsize images from their native size (288 x 432) to the closest legacy video size (300 x 450) – adhering to the fit-to-square rule for the 450 x 450 image size. Native TMS images are significantly closer to 300 x 450. In certain situations, downsizing TMS images to the next lowest legacy video size (160 x 240) can result in significant quality degradation when such downsized images are later displayed in applications or devices.

UI Best Practices for Audio Stream Recognition

The following are recommended best practices for applications that recognize streaming audio. 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 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

© 2000 to present. Gracenote, Inc. All rights reserved.

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.

Provide Clear and Accessible Instructions

Most failed recognitions are due to incorrect operation by the user. Provide clear and concise instructions to help the user correctly operate the application to result in a higher match rate and a better user experience.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.
- Use universal street sign images with written instructions to guide the user.

Provide a 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.

Display a Progress Indicator During Recognition

When listening to audio, the application can receive status updates. The status updates indicate what percentage of the recording is completed. Use this information to display a progress bar (indicator) to notify the user.

Use Animations During Recognition

Display a simple image or animation that shows how to properly perform audio recognition, such as holding the device near the audio source if applicable.

Using Vibration, Tone, or Both to Indicate Recognition Complete

The user may not see visual notifications if they are holding the recording device up to an audio source. Also, 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.

Display Help Messages for Failed Recognitions

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.

© 2000 to present. Gracenote, Inc. All rights reserved.

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 audio recognition.

© 2000 to present. Gracenote, Inc. All rights reserved.

Data Models

The following table links to GNSDK for Mobile data models for the corresponding query response type:

- **Music**
  - GnResponseAlbums
  - GnResponseTracks
  - GnResponseContributors

© 2000 to present. Gracenote, Inc. All rights reserved.

API Reference Documentation

You can develop GNSDK for Mobile applications using several object-oriented languages. The following table describes these APIs and where to find the corresponding API Reference documentation :

API Name	Description	Location in Package
Objective-C API Reference	API descriptions of the GNSDK for Mobile Objective-C interface	docs/Content/api_ref_objc/html/index.html

Playlist PDL Specification

The GNSDK for Mobile Playlist Definition Language (PDL) is a query syntax, similar to Structured Query Language (SQL), that enables flexible custom playlist generation using human-readable text strings. PDL allows developers to dynamically create custom playlists. By storing individual PDL statements, applications can create and manage multiple preset and user playlists for later use.

PDL statements express the playlist definitions an application uses to determine what items are included in resulting playlists. PDL supports logic operators, operator precedence and in-line arithmetic. PDL is based on Search Query Language (SQL). This section assumes you understand SQL and can write SQL statements.

NOTE: Before implementing PDL statement functionality for your application, carefully consider if the provided More Like This function, gnsdk_playlist_generate_morelikethis( ) meets your design requirements. Using the More Like This function eliminates the need to create and validate PDL statements.

PDL Syntax

This topic discusses PDL keywords, operators, literals, attributes, and functions.

Note: Not all keywords support all operators. Use gnsdk_playlist_statement_validate() to check a PDL Statement, which generates an error for invalid operator usage.

Keywords

PDL supports these keywords:

© 2000 to present. Gracenote, Inc. All rights reserved.

Keyword	Description	Required or Optional	PDL Statement Order
GENERATE PLAYLIST	All PDL statements must begin with either GENERATE PLAYLIST or its abbreviation, GENPL	Required	1
WHERE	Specifies the attributes and threshold criteria used to generate the playlist. If a PDL statement does not include the WHERE keyword, Playlist operates on the entire collection.	Optional	2
ORDER	Specifies the criteria used to order the results' display. If a PDL statement does not include the ORDER keyword, Playlist returns results in random order. Example: Display results in based on a calculated similarity value; tracks having greater similarity values to input criteria display higher in the results. The expression format is: <identifier> <operator> <identifier>	Optional	3
LIMIT	Specifies criteria used to restrict the number of returned results. Also uses the keywords RESULT and PER. Example: Limiting the number of tracks displayed in a playlist to 30 results with a maximum of two tracks per artist. The expression format is: <identifier> <operator> <identifier>	Optional	4
SEED	Specifies input data criteria from one or more idents. Typically, a Seed is the This in a More Like This playlist request. Example: Using a Seed of Norah Jones' track Don't Know Why to generate a playlist of female artists of a similar genre. The expression format is: <identifier> <operator> <identifier>	Optional	NA

© 2000 to present. Gracenote, Inc. All rights reserved.

Example: Keyword Syntax

This PDL statement example shows the syntax for each keyword. In addition to <att_imp>, <expr>, and <score> discussed above, this example shows:

- <math_op> is one of the valid PDL mathematical operators.
- <number> is positive value.
- <attr_name> is a valid attribute, either a Gracenote-delivered attribute or an implementation-specific attribute.

```
GENERATE PLAYLIST
 WHERE <att_imp> [<math_op> <score>][ AND|OR <att_imp>]
ORDER <expr>[ <math_op> <expr>]
 LIMIT [number RESULT | PER <attr_name>][,number [ RESULT | PER <attr_name>]]
```

Operators

PDL supports these operators:

Operator Type	Available Operators
Comparison	>, >=, <, <=, ==, !=, LIKE LIKE is for fuzzy matching, best used with strings; see PDL Examples
Logical	AND, OR
Mathematical	+, -, *, /

Literals

PDL supports the use of single (') and double (") quotes, as shown:

- Single quotes: 'value one'
- Double quotes: "value two"
- Single quotes surrounded by double quotes: "'value three'"

You must enclose a literal in quotes or Playlist evaluates it as an attribute.

Attributes

Most attributes require a Gracenote-defined numeric identifier value or GDO Seed.

- Identifier: Gracenote-defined numeric identifier value is typically a 5-digit value; for example, the genre identifier 24045 for Rock. These identifiers are maintained in lists in Gracenote Service; download the lists using GNSDK for Mobile Manager's Lists and List Types APIs
- GDO Seed: Use GNSDK Manager's GDO APIs to access XML strings for Seed input.

© 2000 to present. Gracenote, Inc. All rights reserved.

The following table shows the supported system attributes and their respective required input. The first four attributes are the GOET attributes.

The delivered attributes have the prefix GN_ to denote they are Gracenote standard attributes. You can extend the attribute functionality in your application by implementing custom attributes; however, do not use the prefix GN_.

Name	Attribute	Required Input
Genre Origin Era Artist Type Mood Tempo	GN_Genre GN_Origin GN_Era GN_ArtistType GN_Mood GN_Tempo	Gracenote-defined numeric identifier value GDO Seed XML string
Artist Name Album Name	GN_ArtistName GN_AlbumName	Text string

Functions

PDL supports these functions:

- RAND(max value)
- RAND(max value, seed)

PDL Statements

This topic discusses PDL statements and their components.

Attribute Implementation <att_imp>

A PDL statement is comprised of one or more attribute implementations that contain attributes, operators, and literals. The general statement format is:

"GENERATE PLAYLIST WHERE <attribute> <operator> <criteria>"

You can write attribute implementations in any order, as shown:

GN_ArtistName == "ACDC"

or

"ACDC" == GN_ArtistName

WHERE and ORDER statements can evaluate to a score; for example:

"GENERATE PLAYLIST WHERE LIKE SEED > 500"

WHERE statements that evaluate to a non-zero score determine what idents are in the playlist results. ORDER statements that evaluate to a non-zero score determine how idents display in the playlist results.

© 2000 to present. Gracenote, Inc. All rights reserved.

Expression <expr>

An expression performs a mathematical operation on the score evaluated from a attribute implementation.

[<number> <math_op>] <att_imp>

For example:

3 * (GN_Era LIKE SEED)

Score <score>

Scores can range between -1000 and 1000.

For boolean evaluation, True equals 1000 and False equals 0.

Note: For more complex statement scoring, concatenate attribute implementations and add weights to a PDL statement.

Example: PDL Statements

The following PDL example generates results that have a genre similar to and on the same level as the seed input. For example, if the Seed genre is a Level 2: Classic R&B/Soul, the matching results will include similar Level 2 genres, such as Neo-Soul.

```
"GENERATE PLAYLIST WHERE GN_Genre LIKE SEED"
```

This PDL example generates results that span a 20-year period. Matching results will have an era value from the years 1980 to 2000.

```
"GENERATE PLAYLIST WHERE GN_Era >= 1980 AND GN_Era < 2000"
```

This PDL example performs fuzzy matching with Playlist, by using the term LIKE and enclosing a string value in single (') or double (") quotes (or both, if needed). It generates results where the artist name may be a variation of the term ACDC, such as:

- ACDC
- AC/DC
- AC*DC

```
"GENERATE PLAYLIST WHERE (GN_ArtistName LIKE 'ACDC')"
```

The following PDL example generates results where:

- The tempo value must be less than 120 BPM.
- The ordering displays in descending order value, from greatest to least (119, 118, 117, and so on).
- The genre is similar to the Seed input.

```
"GENERATE PLAYLIST WHERE GN_Tempo > 120 ORDER GN_Genre LIKE SEED"
```

© 2000 to present. Gracenote, Inc. All rights reserved.

GNSDK Glossary

C

Client ID

Each customer receives a unique Client ID string from Gracenote. This string uniquely identifies each application and lets Gracenote deliver the specific features for which the application is licensed. The Client ID string has the following format: 123456-789123456789012312. The first part is a six-digit Client ID, and the second part is a 17-digit Client ID Tag.

Contributor

A Contributor refers to any person who plays a role in an AV Work. Actors, Directors, Producers, Narrators, and Crew are all consider a Contributor. Popular recurring Characters such as Batman, Harry Potter, or Spider-man are also considered Contributors in Video Explore.

Credit

A credit lists the contribution of a person (or occasionally a company, such as a record label) to a recording. Generally, a credit consists of: The name of the person or company. The role the person played on the recording (an instrument played, or another role such as composer or producer). The tracks affected by the contribution. A set of optional notes (such as “plays courtesy of Capitol records”). See Role

F

Features

Term used to encompass a particular media stream's characteristics and attributes; this is metadata and information accessed from processing a media stream. For example, when submitting an Album's Track, this includes information such as fingerprint, mood, and tempo metadata (Attributes).

Fingerprint

Fingerprint Types: Cantametrix Philips microFAPI nanoFAPI

G

Generation Criterion

A selection rule for determining which tracks to add to a playlist.

© 2000 to present. Gracenote, Inc. All rights reserved.

Generator

Short for playlist generator. This is a term used to indicate that an artist has at least one of the extended metadata descriptors populated. The metadata may or may not be complete, and the artist text may or may not be locked or certified. See also: extended metadata, playlist optimized artist.

Genre

A categorization of a musical composition characterized by a particular style.

L

Link Module

A module available in the Desktop and Auto products that allows applications to access and present enriched content related to media that has been identified using identification features.

M

Manual Playlist

A manual playlist is a playlist that the user has created by manually selecting tracks and a play order. An auto-playlist is a playlist generated automatically by software. This distinction only describes the initial creation of the playlist; once created and saved, all that matters to the end-user is whether the playlist is static (and usually editable) or dynamic (and non-editable). All manual playlists are static; the track contents do not change unless the end-user edits the playlist. An auto-playlist may be static or dynamic.

Metadata

Data about data. For example, metadata such as the artist, title, and other information about a piece of digital audio such as a song recording.

Mood

Track-level perceptual descriptor of a piece of music, using emotional terminology that a typical listener might use to describe the audio track; includes hierarchical categories of increasing granularity. See Sonic Attributes.

More Like This

A mechanism for quickly generating playlists from a user's music collection based on their similarity to a designated seed track, album, or artist.

© 2000 to present. Gracenote, Inc. All rights reserved.

Multiple Match

During CD recognition, the case where a single TOC from a CD may have more than one exact match in the MDB. This is quite rare, but can happen. For example with CDs that have only one track, it is possible that two of these one-track CDs may have exactly the same length (the exact same number of frames). There is no way to resolve these cases automatically as there is no other information on the CD to distinguish between them. So the user must be presented with a dialog box to allow a choice between the alternatives. See also: frame, GN media database, TOC.

MusicID Module

Enables MusicID recognition for identifying CDs, digital music files and streaming audio and delivers relevant metadata such as track titles, artist names, album names, and genres. Also provides library organization and direct lookup features.

MusicID-File Module

A feature of the MusicID product that identifies digital music files using a combination of waveform analysis technology, text hints and/or text lookups.

P

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.

Playlist

A set of tracks from a user's music collection, generated according to the criteria and limits defined by a playlist generator.

Popularity

Popularity is a relative value indicating how often metadata for a track, album, artist and so on is accessed when compared to others of the same type. Gracenote's statistical information about the popularity of an album or track, based on aggregate (non-user-specific) lookup history maintained by

© 2000 to present. Gracenote, Inc. All rights reserved.

Gracenote servers. Note that there's a slight difference between track popularity and album popularity statistics. Track popularity information identifies the most popular tracks on an album, based on text lookups. Album popularity identifies the most frequently looked up albums, either locally by the end-user, or globally across all Gracenote users. See Rating and Ranking

R

Ranking

For Playlist, ranking refers to any criteria used to order songs within a playlist. So a playlist definition (playlist generator) may rank songs in terms of rating values, or may rank in terms of some other field, such as last-played date, bit rate, etc.

Rating

Rating is a value assigned by a user for the songs in his or her collection. See Popularity and Ranking.

Role

A role is the musical instrument a contributor plays on a recording. Roles can also be more general, such as composer, producer, or engineer. Gracenote has a specific list of supported roles, and these are broken into role categories, such as string instruments, brass instruments. See Credit.

S

Seed Track, Disc, Artist, Genre

Used by playlist definitions to generate a new playlist of songs that are related in some way, or similar, to a certain artist, album, or track.This is a term used to indicate that an artist has at least one of the extended metadata descriptors populated. The metadata may or may not be complete, and the artist text may or may not be locked or certified. See also: extended metadata, playlist optimized artist, playlist-related terms.

Sonic Attributes

The Gracenote Service provides two metadata fields that describe the sonic attributes of an audio track. These fields, mood and tempo, are track-level descriptors that capture the unique characteristics of a specific recording. Mood is a perceptual descriptor of a piece of music, using emotional terminology that a typical listener might use to describe the audio track. Tempo is a description of the overall perceived speed or pace of the music. The Gracenote mood and

© 2000 to present. Gracenote, Inc. All rights reserved.

tempo descriptor systems include hierarchical categories of increasing granularity, from very broad parent categories to more specific child categories.

T

Target Field

The track attribute used by a generation criterion for selecting tracks to include in a generated playlist.

Tempo

Track-level descriptor of the overall perceived speed or pace of the music; includes hierarchical categories of increasing granularity. See Sonic Attributes.

TOC

Table of Contents. An area on CDs, DVDs, and Blu-ray discs that describes the unique track layout of the disc.

© 2000 to present. Gracenote, Inc. All rights reserved.