



# **GNSDK C**

## Developer Guide

Release: 3.08.6.5437

Published: 9/14/2017 1:07 PM

Copyright 2017 Gracernote, Inc.. All rights reserved.

Information in this document is subject to change without notice. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of those agreements. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Gracernote, Inc..

Gracernote, Inc.

## Contents

<b>Chapter 1 Concepts</b>	<b>1</b>
1.1 About Gracenote	1
1.2 What is the GNSDK?	1
1.3 Gracenote Media Elements	2
1.3.1 Genre and Other List-Dependent Values	2
1.3.1.1 Genres and List Hierarchies	3
1.3.1.2 Simplified and Detailed Hierarchical Groups	5
1.3.2 Core and Enriched Metadata	5
1.3.3 Mood and Tempo (Sonic Attributes)	6
1.3.4 Classical Music Metadata	6
1.3.5 Third-Party Identifiers and Preferred Partners	9
1.4 Music Modules	9
1.4.1 MusicID Overview	10
1.4.1.1 CD TOC Recognition	11
TOC Identification	11
Multiple TOC Matches	12
Multiple TOCs and Fuzzy Matching	12
1.4.1.2 Text-Based Recognition	13
1.4.1.3 Fingerprint-Based Recognition	13
About DSP	14
1.4.2 MusicID-File Overview	14
1.4.2.1 Waveform and Metadata Recognition	14
1.4.2.2 Advanced Processing Methods	15
LibraryID	15

AlbumID .....	16
TrackID .....	16
MusicID-File Best Practices .....	16
Usage Notes .....	17
1.4.2.3 MusicID vs. MusicID-File .....	17
1.4.3 MusicID Stream .....	18
1.4.4 Music Enrichment .....	19
1.4.5 Playlists .....	19
1.4.5.1 Collection Summaries .....	20
1.4.5.2 More Like This .....	20
1.4.5.3 .Playlist Requirements and Recommendations .....	20
Simplified Playlist Implementation .....	20
Playlist Content Requirements .....	21
Playlist Storage Recommendations .....	21
Playlist Resource Requirements .....	21
Playlist Level Equivalency for Hierarchical Attributes .....	22
1.4.5.4 Key Playlist Components .....	22
Media metadata: Metadata of the media items (or files) .....	23
Attributes: Characteristics of a media item, such as Mood or Tempo .....	23
Unique Identifiers .....	23
Collection Summary .....	24
Storage .....	24
Seed .....	24
Playlist generation .....	24
1.4.5.5 Mood Overview .....	25
Mood Descriptors .....	25
Mood Valence/Arousal Model .....	25
Mood Levels .....	26
1.4.5.6 Level 1 Valence/Arousal Map .....	27
1.4.5.7 Level 2 Valence/Arousal Map .....	27
Navigating with Mood .....	27
Slider Navigation .....	28

Grid Navigation .....	28
Bubble Magnitude Navigation .....	29
Other Mood Design Considerations .....	30
Mood Level Arousal/Valance Values .....	31
Level 1 Mood Levels .....	31
Level 2 Mood Levels .....	33
1.4.6 Rhythm Overview .....	37
1.4.7 MusicID-Match Overview .....	39
1.5 Image Formats and Dimensions .....	39
1.5.1 Available Image Dimensions .....	40
1.5.2 Common Media Image Dimensions .....	40
1.5.2.1 Music Cover Art .....	40
1.5.2.2 Artist Images .....	41
1.5.2.3 Genre Images .....	41
1.5.2.4 Video Cover Art .....	41
Video Image Dimension Variations .....	41
Video (AV Work) Images .....	42
Video Product Images .....	42
Video Contributor Images .....	42
1.5.3 Image Best Practices .....	42
1.5.3.1 Using a Default Image When Cover Art Is Missing .....	43
1.5.3.2 Image Resizing Guidelines .....	43
<b>Chapter 2 Setup and Samples .....</b>	<b>45</b>
2.1 Modules Overview .....	45
2.1.1 Modules in the GNSDK Package .....	47
2.1.1.1 GNSDK Modules .....	47
2.2 Using the Sample Applications .....	48
2.2.1 Sample Applications and Reference Applications .....	48
2.2.1.1 Common Flow and Layout .....	48
2.2.1.2 Sample Applications .....	49
2.2.1.3 Reference Applications .....	51
2.2.2 GNSDK C Quick Start Tutorial .....	52
2.2.2.1 Introduction .....	52

2.2.2.2 Setting Up Your Development Environment .....	53
Include Files .....	53
Library Files .....	53
Source Files .....	53
Executing The Application .....	54
2.2.2.3 Coding the App .....	54
Code the Includes, Defines and Local Function Declarations ..	54
Code the main() Function .....	55
Code GNSDK Initializations .....	56
Code the Lookup Function .....	57
Code the Display Function .....	59
Code the Shutdown Function .....	59
2.2.2.4 Conclusion .....	60
2.2.2.5 Reference .....	60
2.2.3 Using the Sample Applications .....	65
2.2.3.1 Sample Applications and Reference Applications .....	65
Common Flow and Layout .....	66
Sample Applications .....	67
Reference Applications .....	69
2.2.4 Building a Sample Application .....	69
2.2.4.1 Build Environment Requirements .....	70
2.2.4.2 Build Steps .....	70
Supported make commands .....	70
2.2.4.3 Including Client and License Information .....	71
2.2.4.4 Directing Output and Log Files .....	71
2.2.4.5 Platform-Specific Build Instructions .....	71
Windows .....	71
Windows CE .....	73
Linux ARM .....	73
Linux and Solaris .....	74
MacOS .....	74
2.2.5 GNSDK C Sample Application Walkthrough .....	74

2.2.5.1 Albums That This Sample Queries On .....	75
2.2.5.2 Prerequisites .....	78
2.2.5.3 Initialization .....	78
Initialize the GNSDK Manager .....	78
Enable SDK Logging .....	79
Initialize Storage and Local Lookup .....	79
Initialize the MusicID Library Manager .....	80
Initialize the User Handle .....	80
Local-Only Registration .....	81
Initialize Localization .....	82
2.2.5.4 Queries .....	82
Album Lookup .....	83
GDO Navigation and Display .....	83
Parsing the Returned Response GDO .....	84
Parsing the Child GDO .....	85
Extract Metadata and Display .....	86
Navigate and Parse Additional Child GDOs .....	86
2.2.5.5 Releasing Resources and Shutting Down .....	86
<b>Chapter 3 Develop and Implement .....</b>	<b>89</b>
3.1 Application Flow .....	89
3.2 Working with Header Files and Libraries .....	90
3.3 Authorizing GNSDK Applications .....	91
3.3.1 License Files .....	92
3.3.2 Client IDs .....	92
3.3.3 Users .....	93
3.4 Initializing and Shutting Down GNSDK .....	95
3.4.1 Initializing an Application .....	95
3.4.1.1 Specifying the License File .....	96
3.4.2 Shutting Down an Application .....	96
3.4.3 Example: Initializing and Shutting Down .....	97
3.5 Setting Network and Proxy Options .....	97

3.5.1 Proxy Server Options .....	97
3.5.2 .....	99
3.5.3 Network Options .....	99
3.5.4 Testing Connection Ability .....	99
3.6 Configuring Logging .....	100
3.6.1 Enabling GNSDK Logging .....	100
3.6.2 Adding to GNSDK Logs .....	101
3.6.3 Implementing Callback Logging .....	102
3.7 Implementing Status Callbacks .....	104
3.7.1 Displaying Operation Status .....	104
3.7.2 Registering a Status Callback .....	105
3.7.3 canceling Operations .....	106
3.7.4 Example: Using a Callback .....	106
3.8 Using Locales .....	106
3.8.1 Loading a Locale .....	106
3.8.1.1 Default Regions and Descriptors .....	108
3.8.2 Locale Groups .....	109
3.8.3 Locale-Dependent Values and List Types .....	110
3.8.3.1 Locale-Dependent Genre Levels .....	110
3.8.3.2 Music Locale-Dependent Values and List Types .....	111
3.8.3.3 Video Locale-Dependent Values .....	112
3.8.3.4 EPG Locale-Dependent Values .....	114
3.8.3.5 Multi-Threaded Access .....	114
3.8.4 Updating Locales and Lists .....	115
3.8.5 Best Practices .....	116
3.8.6 Example: Loading a Locale and Retrieving Locale-Sensitive Metadata .....	117
3.8.7 Using Lists .....	117
3.8.7.1 List and Locale Interdependence .....	118
3.8.7.2 Retrieving Locale Information .....	120
3.8.7.3 Updating Lists .....	121
3.8.7.4 Example: Accessing a Music Genre List .....	121



3.9 Local Storage .....	122
3.9.1 Setting Local and Online Lookup Modes .....	122
3.9.1.1 Lookup Providers .....	122
3.9.1.2 Storage Providers .....	123
3.9.1.3 Lookup Modes .....	123
3.9.1.4 Local and Online Lookup Comparison .....	124
3.9.2 Working with Local Databases .....	126
3.9.2.1 Initializing a Local Database .....	126
3.9.2.2 Setting Local Lookup Mode .....	127
3.9.2.3 Getting Manifest Information about Local Databases .....	128
3.9.2.4 Using Results Databases .....	129
Application Flow .....	130
Managing Disk Space .....	130
3.9.3 Querying and Updating a Results Database .....	131
Querying and Updating a Results Database .....	131
Results Database Functions .....	132
Examples: Results Databases .....	133
3.9.3.1 Using SQLite for Storage and Caching .....	133
Setting Location, Access, and File Size .....	134
Linking to External SQLite Libraries .....	136
3.9.4 Load Balancing .....	137
3.9.4.1 Implementation Considerations .....	137
3.10 About Gracenote Data Objects (GDOs) .....	138
3.10.1 GDO Types .....	139
3.10.2 Response GDOs and Child GDOs .....	139
3.10.3 Child Keys and Value Keys .....	140
3.10.4 Full and Partial Metadata Results .....	140
3.10.4.1 Online and Local Database Queries .....	141
3.10.5 Matches That Require Decisions .....	141
3.10.5.1 About the Text Match Score .....	142
3.10.5.2 About Video Product TOC Matches .....	142
3.10.6 GDO Workflows .....	143

3.10.6.1 GDO Workflow for a Single GDO Match .....	143
GDO Workflow Steps for Album Title Text Lookup - Single GDO Response .....	145
3.10.6.2 GDO Workflow for Multiple GDO Matches .....	146
GDO Workflow Steps for Album Title Text Lookup - Multiple GDO Response .....	148
3.10.7 Common GDO Tasks .....	149
3.10.7.1 Retrieving a Track GDO from an Album GDO .....	149
3.10.7.2 Getting the Type for a GDO .....	151
3.10.7.3 Checking for Full and Partial Results .....	152
3.10.7.4 Checking Whether a Match Needs a Decision .....	153
3.10.7.5 Serializing a GDO .....	154
3.10.7.6 Rendering a GDO as XML .....	155
3.10.8 GDO Navigation Examples .....	155
3.10.8.1 Example: Looking Up an Album by a TOC .....	155
3.10.8.2 Example: Accessing Album and Track Metadata Using Album GDO .....	156
3.10.9 Working with Non-GDO Identifiers .....	159
3.10.9.1 Examples of Non-GDO Identifiers .....	161
3.10.9.2 GNSDK Identifiers Comparison Matrix .....	161
3.10.9.3 Converting Non-GDO Identifiers to GDOs .....	162
Example: Creating a GDO from an Album TUI .....	162
Example: Creating a GDO from a Track TagID .....	162
Example: Creating a GDO from an Video Product ID .....	163
Example: Creating a GDO from a CDDDBID .....	163
Example: Creating a GDO from Raw CDDDBID .....	163
3.10.10 Rendering a GDO as XML .....	163
3.10.10.1 Example: Rendering a GDO as XML .....	165
3.10.11 Creating a GDO From XML .....	165
3.11 Identifying Music .....	165
3.11.1 Identifying Music Using a CD TOC .....	166
3.11.1.1 Example: Identifying an Album Using a CD TOC .....	166

3.11.2 Identifying Music Using Text .....	167
3.11.2.1 Creating and Executing a Query for a Text-based Lookup .....	167
3.11.2.2 Processing Text-based Lookup Results .....	168
3.11.2.3 Example: Text Lookup for a Track .....	169
3.11.2.4 Best Practices for MusicID Text Match Queries .....	172
3.11.2.5 Identifying Music Using Batch Processing .....	174
Creating a Batch Query .....	174
Performing a Batch Query and Retrieving Data .....	176
Setting Batch Lookup Options .....	177
Examining Batch Functions .....	177
Example: Executing Batch Lookups .....	178
3.11.3 Using MusicID-File .....	179
3.11.3.1 TrackID and AlbumID Implementation Steps .....	179
3.11.3.2 LibraryID Implementation Steps .....	179
3.11.3.3 Initialization .....	180
3.11.3.4 Creating a MusicID-File Query Handle .....	180
TrackID and AlbumID .....	180
LibraryID .....	180
3.11.3.5 Creating FileInfo Objects .....	181
3.11.3.6 Setting MusicID-File Query Options .....	182
3.11.3.7 Setting Media File Identification Process .....	182
Fingerprinting .....	183
Setting and Getting FileInfo Metadata .....	184
3.11.3.8 Performing the Identification Query .....	184
3.11.3.9 Getting Results with TrackID and AlbumID .....	185
3.11.3.10 Getting Results with LibraryID .....	187
3.11.3.11 Releasing Resources and Shutting Down .....	188
3.11.3.12 Advanced Music Identification using MusicID-File .....	188
TrackID .....	188
Example: Implementing TrackID .....	188
AlbumID .....	191
Example: Implementing AlbumID .....	191

LibraryID .....	194
Example: Implementing LibraryID .....	194
3.11.4 Identifying Streaming Music .....	196
3.11.4.1 MusicID Stream Callbacks .....	197
3.11.4.2 Example: Identifying Streaming Music .....	198
3.11.5 Implementing Audio Suitability Processing (ASP) .....	199
3.11.5.1 Setting ASP for Queries .....	199
3.11.5.2 Setting ASP for MusicID Stream .....	199
3.11.6 UI Best Practices for Audio Stream Recognition .....	200
3.11.6.1 Provide Clear and Accessible Instructions .....	201
3.11.6.2 Provide a Demo Animation .....	201
3.11.6.3 Display a Progress Indicator During Recognition .....	202
3.11.6.4 Use Animations During Recognition .....	202
3.11.6.5 Using Vibration, Tone, or Both to Indicate Recognition Complete .....	202
3.11.6.6 Display Help Messages for Failed Recognitions .....	202
3.11.6.7 Allow the User to Provide Feedback .....	202
3.11.7 Identifying Music Using Batch Processing .....	202
3.11.7.1 Creating a Batch Query .....	203
3.11.7.2 Performing a Batch Query and Retrieving Data .....	204
3.11.7.3 Setting Batch Lookup Options .....	205
3.11.7.4 Examining Batch Functions .....	206
3.11.7.5 Example: Executing Batch Lookups .....	207
3.11.8 Collaborative Artists Best Practices .....	207
3.11.8.1 Handling Collaborations when Processing a Collection .....	208
3.11.8.2 Displaying Collaborations during Playback .....	208
3.11.8.3 Displaying Collaborations in Navigation .....	209
3.11.8.4 Handling Collaborations in Playlists .....	210
3.11.9 Navigating Music GDOs .....	210
3.11.9.1 Accessing Classical Music Metadata .....	212
Classical Album Example .....	212
3.11.9.2 Accessing External IDs .....	215
Example .....	216

3.11.9.3 Accessing Collaborative Artists Metadata .....	218
Navigating Collaborations .....	218
Working with Collaborative Artists in Text-Lookup Results ....	219
Using the GNSDK_GDO_VALUE_COLLABORATOR_	
RESULT Key .....	220
Collaborative Artist Input Scenarios .....	221
3.11.9.4 Input Collaboration Found .....	221
3.11.9.5 Input Collaboration Not Found .....	221
3.11.9.6 Input is Not a Collaboration .....	222
3.11.9.7 Improving Matches Using Both CD TOC and	
Fingerprints .....	222
Using a TOC and an Existing Fingerprint .....	223
Using a TOC and a Generated Fingerprint .....	223
3.11.9.8 Accessing Enriched Content using Asset Fetch .....	223
Using the Asset Fetch API .....	224
Example: Accessing Album Cover Art .....	225
3.11.9.9 Accessing Enriched Content using Link .....	225
Using the Link Module .....	226
Retrieving Cover Art and Artist Images .....	226
Retrieving Genre Art .....	228
Retrieving Genre Art with a GDO .....	228
Retrieving Genre Art without a GDO .....	228
Setting Image Size Retrieval Order .....	229
Example: Accessing Album Cover Art .....	229
3.11.9.10 Using Enriched Content URLs .....	230
3.11.9.11 Music Metadata Reference Overview .....	231
Album Metadata Example .....	231
3.11.9.12 Generating a Playlist .....	234
Creating a Collection Summary .....	235
Populating a Collection Summary .....	235
Enabling Playlist Attributes .....	235
Adding Data to a Collection Summary .....	236
How Playlist Gathers Data .....	237

Adding List Elements to Collection Summaries .....	237
Working with Local Storage .....	238
Generating a Playlist Using More Like This .....	238
Accessing Playlist Results .....	240
Working with Multiple Collection Summaries .....	241
Join Performance and Best Practices .....	242
Synchronizing Collection Summaries .....	242
Iterating the Physical Media .....	243
Processing the Collection .....	243
Example: Implementing a Playlist .....	243
Implementing Mood .....	249
Initializing the Mood Module .....	249
Enumerating Data Sources using Mood Providers .....	250
Creating and Populating a Mood Presentation .....	251
Filtering Mood Results .....	252
Shutting Down Mood .....	252
Example: Working with Mood .....	252
Accessing Mood and Tempo Metadata .....	255
3.12 Mood .....	255
3.13 Tempo .....	256
Playlist PDL Specification .....	258
PDL Syntax .....	259
3.13.1 Keywords .....	259
3.13.1.1 Example: Keyword Syntax .....	261
3.13.2 Operators .....	261
3.13.3 Literals .....	262
3.13.4 Attributes .....	262
3.13.5 Functions .....	263
3.13.6 PDL Statements .....	263
3.13.6.1 Attribute Implementation <att_imp> .....	263
3.13.6.2 Expression <expr> .....	264
3.13.6.3 Score <score> .....	264
Example: PDL Statements .....	264
3.13.6.4 Implementing Rhythm .....	265

Initializing the Rhythm Module .....	266
Creating a Rhythm Query .....	266
Creating Recommendations For Queries and Radio Station Playlists .....	267
Providing Feedback .....	267
Tuning Playlists .....	269
Saving Radio Stations .....	270
Shutting Down Rhythm .....	270
Sample Application .....	271
3.13.6.5 Deploying Android Applications .....	271
GNSDK Android Permissions .....	271
3.14 Testing an Application .....	272
3.14.1 Enabling Test Mode .....	272
3.14.2 Setting Environment Variables .....	272
3.14.3 Enabling Full Access to Gracenote Service .....	273
3.15 Submitting New and Updated Content .....	273
3.15.1 Submit Terminology .....	274
3.15.2 Submit Process .....	274
3.15.3 Submit APIs .....	275
3.15.4 Editable GDO Child and Value Keys .....	276
3.15.4.1 Editable GDO Considerations .....	278
3.15.5 Submitting Album Metadata .....	279
3.15.5.1 Example: Editing and Submitting an Existing Album GDO .....	281
3.15.6 Submitting Track Features .....	281
3.15.6.1 Example: Submitting Track Features (1) .....	282
3.15.6.2 Example: Submitting Track Features (2) .....	282
3.15.7 Submitting Parcels .....	283
3.15.7.1 Example: Submitting an Album Parcel .....	284
3.15.8 Availability of Edited Data Cache .....	284
3.15.9 Synchronizing Dependent Fields .....	285
3.15.10 Enabling Test Mode for Submit Finalization .....	285

3.0.1 API Reference Overview .....	288
<b>Chapter 4 Data Models .....</b>	<b>289</b>
4.1 C Data Model .....	289
<b>Chapter 5 API Reference .....</b>	<b>292</b>
5.1 Playlist PDL Specification .....	292
5.1.1 PDL Syntax .....	292
5.1.1.1 Keywords .....	293
Example: Keyword Syntax .....	294
5.1.1.2 Operators .....	295
5.1.1.3 Literals .....	295
5.1.1.4 Attributes .....	295
5.1.1.5 Functions .....	296
5.1.1.6 PDL Statements .....	296
Attribute Implementation <att_imp> .....	296
Expression <expr> .....	297
Score <score> .....	297
5.1.2 Example: PDL Statements .....	297
<b>Chapter 5 Using Docs and Resources .....</b>	<b>298</b>
5.2 Searching .....	298
5.3 Using the Toolbar .....	299
5.4 Components and Resources .....	299
<b>Glossary .....</b>	<b>302</b>
<b>Index .....</b>	<b>308</b>





## Chapter 1 Concepts

This section introduces GNSDK, and presents important product concepts.

### 1.1 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](http://www.gracenote.com).

### 1.2 What is the GNSDK?

Gracenote SDK (GNSDK) is a platform that delivers Gracenote technologies to devices, desktop applications, web sites, and back-end 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 multi-threaded environments. GNSDK is also lightweight—made to run in desktop applications and even to support popular portable devices.

## 1.3 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:

### Music

- Music CD
- Album
- Track
- Artist
- Contributor

### Video

- Video Product (DVD/Blu-Ray)
- AV Work
- Contributor
- Series
- Season

### 1.3.1 Genre and Other List-Dependent Values

GNSDK 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, and roles.

List-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. These kinds of list-based metadata values are called *locale-dependent*.

### 1.3.1.1 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.

### 1.3.1.2 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.

Below are examples of a Simplified and Detailed Genre Hierarchy Groups. The values below are for documentation purposes only. Please contact your Gracenote representative for more information.

#### 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

Contact your Gracenote representative for more detail.

### 1.3.2 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.

For music, core metadata for albums, tracks, and artists includes:

- Genres
- Origins
- Era
- Title
- Types
- External IDs

Enriched metadata includes album cover art, artist images, and more.

### **1.3.3** *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.

To use this feature, your application requires special metadata entitlements. Contact your Gracenote representative for more information.

### **1.3.4** *Classical Music Metadata*



Gracenote also supports classical music metadata, which is typically more complex than non-classical music. Gracenote uses a Three-Line Solution (TLS)

to map classical metadata to an AUDIO\_WORK Track, Artist, and Album media elements. The tables below show this mapping.

### Classical Music Three-Line Solution Metadata Mapping

Retrieved AUDIO_WORK Element	Returned Classical Music Metadata
Track	<p>[Composer Short Name]: [Work Title] In {Key}, {Opus}, {Cat#}, {"Nickname"} - [Movement#]. [M. Name]</p> <p>Example:</p> <p>Dvořák: Symphony #9 In E Minor, Op. 95, B 178, "From The New World" - 1. Adagio</p> <p>Dvořák [Composer Short Name]: Symphony #9 [Work Title] In E Minor {Key}, Op. 95 {Opus}, B 178 {Catalog #}, "From The New World" {Nickname} - 1. Adagio [Movement #] [M Name]</p>
Artist	<p>{Soloist(s)}, {Conductor}; {Ensemble #1}, {Ensemble #2, Etc.}</p> <p>Example:</p> <p>Anton Dermota, Cesare Siepi, Etc.; Josef Krips: Vienna Philharmonic Orchestra, Vienna State Opera Chorus</p> <p>Anton Dermota (Soloist), Cesare Siepi (Soloist), Etc. (indicates additional soloists)); Josef Krips (Conductor): Vienna Philharmonic Orchestra (Ensemble #1), Vienna State Opera Chorus (Ensemble #2)</p>
Album	<p>Name printed on the spine of the physical CD, if available.</p> <p>Example: Music Of The Gothic Era [Disc 1]</p>

### AUDIO\_WORK Metadata Definitions

Album Name	<p>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.</p> <ul style="list-style-type: none"> <li> General title example: Beethoven: Violin Concerto</li> <li> Formal title example: The Best Of Baroque Music</li> </ul>
------------	---



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. Example: Hilary Hahn; David Zinman: Baltimore Symphony Orchestra
Catalog #	The chronological number of a work as designated by a generally acknowledged scholar of a composer's work if it is a formal part of the Work Title. This is usually comprised of an upper-case letter and a number. The letter indicates either the last initial of the scholar/compiler of the composer's catalog. For Mozart, this would be "K" for Ludwig Ritter von Köchel) or the title of the scholar's catalog of the composer's works (for Bach, this would be "BWV" for "Bach-Werke-Verzeichnis (Bach Works List)."
Composer	The composer(s) that are featured on an album are listed by their last name in the album title (where applicable)
Key	The key signature of the work. It is included in a TLS data string if it is a formal part of the Work Title, such as Beethoven: Violin Concerto in D.  If it is not part of the title (Debussy: Syrinx), it is not included in the data string. TLS exclusively uses the traditional key designations (A, B, C, D, etc.) as opposed to other versions (H-Moll, etc.)
Movement	A constituent part or subsection of a work. A movement can be contained within a single track, or can comprise multiple tracks, but will always share the formal title of the parent work. Movements can, themselves, be subdivided into separate sections delineated by Act, Scene, Part, Variation, etc.
Movement #	The formally-sequenced number of a subsection of a work.
Nickname	This is an alternative title for a formal work that is used synonymously with that work. It is included in a TLS string if it is a formal part of the Work Title. For example: Beethoven: Symphony #9 In D Minor, Op. 125, "Choral"
Opus	Abbreviated as "Op." in the TLS string, this is the publishing number of the work in the composer's canon and is included in the data string if it is a formal part of the Work Title. For example: Strauss (R): Symphony In F Minor, Op. 12
Track	A distinct recorded performance, delineated by a specific time duration that can be quantified in a Table of Contents (TOC).
Track Name	A consistent format 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

Work Title	<p>For TLS purposes, the formal title of a musical work with the inclusion of the composer's Short Name prepended. Work titles come in two versions:</p> <ul style="list-style-type: none"> <li>• “text” title comprising a word or set of words (“The Rite Of Spring”), or</li> <li>• title based on the musical form in which it was composed (Symphony #9, Sonata #1, etc.)</li> </ul> <p>When applicable, a work title can also include work number, key signature, instrumental attribution, catalog number, opus number and/or nickname.</p> <p>Additional information can be included in parentheses, such as The Firebird (1911 Version), in the work title string. Modifiers such as Book, Suite, Vol., Part, etc. are also included in the title string.</p>
------------	--

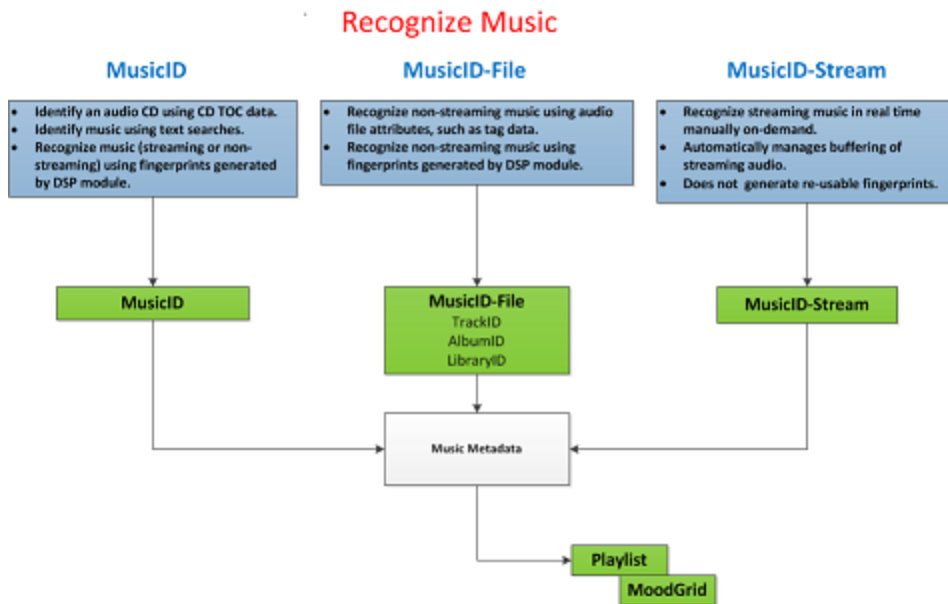
### 1.3.5 Third-Party Identifiers and Preferred Partners

Link can match identified media with third-party identifiers. This allows applications to match media to IDs in stores and other online services—facilitating transactions by helping connect queries directly to commerce.

Gracenote has preferred partnerships with several partners and matches preferred partner content IDs to Gracenote media IDs. Entitled applications can retrieve IDs for preferred partners through Link.

## 1.4 Music Modules

The following diagram shows the kinds of identification queries each music module supports.



## Access Metadata

### 1.4.1 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:

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

- Music Enrichment content, including cover art, artist images, biographies, and reviews

GNSDK 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)

#### 1.4.1.1 CD TOC Recognition

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.

##### TOC Identification

The only information that is guaranteed to be on every standard audio CD is a Table of Contents, or TOC. This is a header at the beginning of the disc giving the precise starting location of each track on the CD, so that CD players can locate the tracks and compute the track length information for their display panels.

This information is given in frames, where each frame is 1/75 of a second. Because this number is so precise, it is relatively unlikely that two unrelated CDs would have the same TOC. This lets Gracenote use the TOC as a relatively unique identifier.

The example below shows a typical TOC for a CD containing 13 tracks:

```
150 26670 52757 74145 95335 117690 144300 163992 188662 209375 231320
253150 281555 337792
```

The first 13 numbers represent the frames from the beginning of the disc that indicate the starting locations of the 13 tracks. The last number is the offset of the lead out, which marks the end of the CD program area.

### Multiple TOC Matches

An album will often have numerous matching TOCs in the Gracenote database. This is because of CD manufacturing differences. More popular discs tend to have more TOCs. Gracenote maintains a catalog of multiple TOCs for many CDs, providing more reliable matching.

The following is an example of multiple TOCs for a single CD album. This particular album has 22 popular TOCs and many other less popular TOCs.

150 26670 52757 74145 95335 117690 144300 163992 188662 209375 231320 253150 281555 337642
150 26670 52757 74145 95335 117690 144300 163992 188662 209375 231320 253150 281555 337792
182 26702 52790 74177 95367 117722 144332 164035 188695 209407 231362 253182 281587 337675
150 26524 52466 73860 94904 117037 143501 162982 187496 208138 230023 251697 279880 335850

### Multiple TOCs and Fuzzy Matching

Gracenote MusicID utilizes several methods to perform TOC matches. This combination of matching methods allows client applications to accurately recognize media in a variety of situations.

- Exact Match - when there is only one Product match for a queried CD TOC
- Multi-Exact Match - when there are multiple Product matches for a queried CD TOC
- Fuzzy Match - allows identification of media that has slight known and acceptable variations from well-recognized media.

### 1.4.1.2 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:

- 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.

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. Also see [Identifying Music Using Text](#) and [Best Practices for MusicID Text Match Queries](#).

### 1.4.1.3 Fingerprint-Based Recognition

Gracenote uses audio fingerprinting as one method to identify tracks. Fingerprints can be generated from audio files and variety of audio sources, including recorded and degraded sources such as radios and televisions. This enables music identification using arbitrary audio sources—including sampling music via mobile devices.

Your application can retain fingerprints for a collection of audio files so they can be used later in queries. For example, your application can fingerprint an entire collection of files in a background thread and reuse them later.

## ***About DSP***

The DSP module is an internal module that provides Digital Signal Processing functionality used by other GNSDK modules. This module is optional unless the application performs music identification or generates audio features for submission to Gracenote.

### **1.4.2 MusicID-File Overview**

MusicID-File provides advanced file-based identification features not included in the MusicID module. MusicID-File can perform recognition using individual 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.

At a high level, MusicID-File APIs implement the following services:

- Identification through waveform fingerprinting and metadata
- Advanced processing methods for identifying individual tracks or file groupings and collections
- Result and status management

MusicID-File can be used with a local database, but it only performs text-matching locally. Fingerprints are not matched locally.

MusicID-File queries never return partial results. They always return full results.

#### **1.4.2.1 Waveform and Metadata Recognition**

The MusicID-File module utilizes both audio data and existing metadata from individual media files to produce the most accurate identification possible.

Your application needs to provide raw, decoded audio data (pulse-code modulated data) to MusicID-File, which processes it to retrieve a unique audio fingerprint. The application can also provide any metadata available for the media file, such as file tags, filename, and perhaps any application metadata. MusicID-

File can use a combination of fingerprint and text lookups to determine a best-fit match for the given data.

The MusicID module also provides basic file-based media recognition using only audio fingerprints. The MusicID-File module is preferred for file-based media recognition, however, as its advanced recognition process provides significantly more accurate results.

### 1.4.2.2 Advanced Processing Methods

The MusicID-File module provides APIs that enable advanced music identification and organization. These APIs are grouped into the following three general categories - LibraryID, AlbumID, and TrackID.

#### LibraryID

LibraryID identifies the best album(s) for a large collection of tracks. It takes into account a number of factors, including metadata, location, and other submitted files when returning results. In addition, it automatically batches AlbumID calls to avoid overwhelming device and network resources.

Normal processing is 100-200 files at a time. 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. If the number of files in a batch exceeds batch size, it will attempt to make an intelligent decision about where to break based on other factors.

All processing in LibraryID is done through callbacks (for example, fingerprinting, setting metadata, returned statuses, returned results, and so on.). The status or result callbacks provide the only mechanism for accessing Response GDOs.



**Note:** For object-oriented applications, for example, those written in C++, GDOs are the same thing as GnDataObjects. All object-oriented response objects are derived from GnDataObject.



## AlbumID

AlbumID identifies the best album(s) for a group of tracks. For example, while the best match for a track would normally be the album where it originally appeared, if the submitted media files as a group are all tracks on a compilation album, then that is identified as the best match. All submitted files are viewed as a single group, regardless of location.

AlbumID assumes submitted tracks are related by a common artist or common album. Your application must be careful to only submit files it believes are related in this way. If your application cannot perform such grouping use LibraryID which performs such grouping internally

## 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. Use TrackID if the original album a track appears on is the best or first result you want to see returned before any compilation, soundtrack, or greatest hits album the track also appears on.

## MusicID-File Best Practices

- Use LibraryID for most applications. LibraryID is designed to identify a large number of audio files. It gathers the file metadata and then groups the files using tag data. LibraryID can only return a single, best match
- Use TrackID or AlbumID if you want all possible results for a track. You can request AlbumID and TrackID to return a single, best album match, or all possible matches. .
- Use AlbumID if your tracks are already pretty well organized by album. For memory and performance reasons, you should only provide a small number of related tracks. Your application should pre-group the audio files, and submit those groups one at a time.
- Use TrackID for one off track identifications and if the original album that a track appears on is the best or first result you want to see returned. TrackID

is best for identifying outliers, that is those tracks unable to be grouped by the application for use with AlbumID. You can provide many files at once, but the memory consumed is directly proportional to the number of files provided. o To keep memory down you should submit a small number of files at a time

## Usage Notes

- As stated above, TrackID and AlbumID are not designed for large sets of submitted files (more than an album's worth). Doing this could result in excessive memory use.
- For all three ID methods, you need to add files for processing manually, one-at-a-time. You cannot add all the files in a folder, volume, or drive in a single call.

### 1.4.2.3 MusicID vs. MusicID-File

Deciding whether to use the MusicID or MusicID-File SDK depends upon whether you are doing a "straightforward lookup" or "media recognition."

Use the MusicID SDK to perform a straightforward lookup. A lookup is considered straightforward if the application has a single type of data and would like to retrieve the Gracenote results for it. The source of the data does not matter (for example, the data might have been retrieved at a different time or from various sources). Examples of straightforward lookups are:

- Doing a lookup with text data only
- Doing a lookup with an audio fingerprint only
- Doing a lookup with a CD TOC
- Doing a lookup with a GDO value. Note that for object-oriented applications, for example, those written in C++, GDOs are the same thing as GnDataObjects. All object-oriented response objects are derived from GnDataObject.

Each of the queries above are completely independent of each other. The data doesn't have to come from actual media (for example, text data could come from user input). They are simply queries with a single, specific input.

Use the MusicID-File SDK to perform media recognition. MusicID-File performs recognition by using a combination of inputs. It assumes that the inputs are from actual media and uses this assumption to determine relationships between the input data. This SDK performs multiple straightforward lookups for a single piece of media and performs further heuristics on those results to arrive at more authoritative results. MusicID-File is capable of looking at other media being recognized to help identify results (for example, AlbumID).



**Note:** If you only have a single piece of input, use MusicID. It is easier to use than MusicID-File, and for single inputs MusicID and MusicID-File will generate the same results.

### **1.4.3** *MusicID Stream*

You can use MusicID Stream to recognize music delivered as a continuous stream. Specifically, MusicID Stream enables these features:

- Recognizes streaming music in real time, on-demand (user-initiated).
- 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. 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, the system waits until enough audio is received. The identification process spawns a thread and completes asynchronously.

Your application can identify audio using Gracenote Service online database or a local fingerprint database. The default behavior attempts a local database match. If this fails, your application can attempt an online database match.

#### **1.4.4 Music Enrichment**

Link provides access to Gracenote Music Enrichment—a single-source solution for enriched content including cover art, artist images, biographies, and reviews. Link and Music Enrichment allow applications to offer enriched user experiences by providing high quality images and information to complement media.

Gracenote provides a large library of enriched content and is the only provider of fully licensed cover art, including a growing selection of international cover art. Music Enrichment cover art and artist images are provided by high quality sources that include all the major record labels.

Examples of enriched content are:

- Album cover art
- Artist images
- Album reviews
- Artist biographies

#### **1.4.5 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 to generate highly relevant playlists.

### 1.4.5.1 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.

### 1.4.5.2 More Like This

More Like This is a powerful and popular navigation feature made possible by Gracenote and GNSDK 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.

### 1.4.5.3 .Playlist Requirements and Recommendations

This topic discusses requirements and recommendations for your Playlist implementation.

#### 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's MusicID or MusicID-File (or both) to recognize music media and create valid GDOs. Note that for object-oriented applications, for example, those written in C++, GDOs are the same thing as GnDataObjects. All object-oriented response objects are derived from GnDataObject.
- 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 23*.

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 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	GNSDK user Mobile device user	Generally only one	Once, normally at start-up	Minimal-to-average, especially as data is ingested only once or infrequently
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 6*. 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.

#### 1.4.5.4 Key Playlist Components

Playlist operates on several key components. The GNSDK 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 operation, and consequently must be implemented with one or both of GNSDK'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).

### 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.



**Note:** For object-oriented applications, for example, those written in C++, GDOs are the same thing as GnDataObjects. All object-oriented response objects are derived from GnDataObject.

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 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 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 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.

#### 1.4.5.5 Mood Overview

Gracenote provides track-based mood data that allow users to generate playlists based on the mood they want to listen to. When the user selects a mood, the application provides a playlist of music that corresponds to the selected mood.

An important advantage of Mood is that it is track-specific and independent of other track metadata. This enables you to create intuitive user interfaces that can create mood-based playlists across different genres, eras, or artist types, and so on. The user can also filter the playlist using one or more of these other attributes to refine the playlist.

---

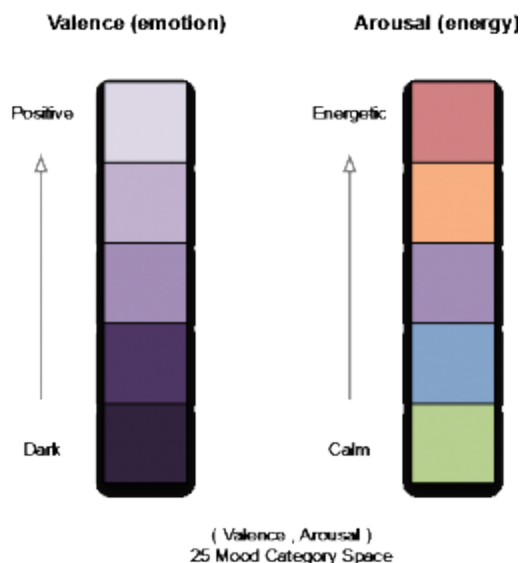
#### Mood Descriptors

MusicID, MusicID File, and MusicID Stream return mood metadata in track results. Gracenote defines over 100 mood types. This granularity of moods is useful in non-automotive applications, such as desktop and tablet interfaces. To simplify mood selection for with limited size interfaces, Gracenote provides a superset of 25 moods called Level 1. Examples of Level 1 range from Peaceful to Excited and from Somber to Aggressive.

#### *Mood Valence/Arousal Model*

Gracenote characterizes moods using a Valence/Arousal model. The mood of every track in the Gracenote repository is expressed as a coupled value of (Valence/Arousal):

- Valence (Emotion) is a psychological term for defining the positivity or negativity of emotions. Valence describes the attractiveness (positive valence) or averseness (negative valence) of an event, object, or situation. For example, the emotions popularly referred to as negative, such as anger and fear, have negative valence. Emotions popularly referred to as positive, such as joy and peacefulness, have positive valence.
- Arousal (Energy) is a psychological term for describing the energy associated with an emotion. For example, the emotions associated with peaceful are considered to have low arousal, while the emotions associated with celebratory have a high arousal.



### ***Mood Levels***

The following tables list mood levels for Level 1 and Level 2 categories. In these examples, the top row of numbers represents Arousal (Energy) values, and the column of numbers on the left represent the Valence (Emotion). For example in Level 1: Peaceful is (1,5), indicating a low Arousal of 1, and high Valence of 5. On the contrary, Aggressive is (5,1) indicating high Arousal of 5, and a low Valence of 1.

For a list of Valence/Arousal value mappings for each mood level, see [Mood Level Arousal/Valence Values](#).

### 1.4.5.6 Level 1 Valence/Arousal Map

L1 (35) Category Sonic Mood Grid

		Arousal →				
		1	2	3	4	5
Valence ↓	1	Somber	Gloomy	Serious	Brooding	Aggressive
	2	Melancholy	Eerie	Yearning	Urgent	Defiant
	3	Sentimental	Supersaturated	Sensual	Fairy	Emerging
	4	Tender	Romantic	Empowering	Swing	Rowdy
	5	Peaceful	Easygoing	Upbeat	Lively	Excited

### 1.4.5.7 Level 2 Valence/Arousal Map

L2 (100) Category Sonic Mood Grid

		Arousal →				
		1	2	3	4	5
Valence ↓	1	Dark Cosmic	Creepy / Ominous	Depressed / Lonely	Gloomy / Soulful	Serious / Cerebral
	2	Solemn / Spiritual	Enigmatic / Mysterious	Sober / Determined	Strumming Yearning	Melodramatic
	3	Wistful / Forlorn	Sad / Soulful	Cool Confidence	Dark Groovy	Sensitive / Exploring
	4	Mysterious / Dreamy	Light Melancholy	Casual Groove	Wary / Defiant	Bittersweet Pop
	5	Lyrical Sentimental	Cool Melancholy	Intimate Bittersweet	Smoky / Romantic	Dreamy Pulse
	6	Tender / Sincere	Gentle Bittersweet	Subtle / Sultry	Dark Playful	Soft Soulful
	7	Romantic / Lyrical	Light Groovy	Dramatic / Romantic	Lush / Romantic	Dramatic Emotion
	8	Refined / Mannered	Awakening / Statly	Sweet / Sincere	Heartfelt Passion	Strong / Stable
	9	Reverent / Healing	Quiet / Introspective	Friendly	Charming / Easygoing	Soulful / Easygoing
	10	Pastoral / Serene	Delicate / Tranquil	Hopeful / Breezy	Cheerful / Playful	Carefree Pop
Valence ↓	6	Thrilling	Dreamy Brooding	Alienated / Brooding	Chaotic / Intense	Aggressive Power
	7	Hypnotic Rhythm	Evocative / Intriguing	Energetic Melancholy	Dark Hard Beat	Heavy Triumphant
	8	Energetic Dreamy	Dark Urgent	Energetic Anxious	Attitude / Defiant	Hard Dark Excitement
	9	Energetic Yearning	Dark Pop	Dark Pop Intensity	Heavy Brooding	Hard Positive Excitement
	10	Intimate	Passionate Rhythm	Energetic Abstract Groove	Edgy / Sexy	Abstract Beat
	1	Sensual Groove	Dark Sparkling Lyrical	Fiery Groove	Arousing Groove	Heavy Beat
	2	Idealistic / Stirring	Focused Sparkling	Triumphant / Rousing	Confident / Tough	Driving Dark Groove
	3	Powerful / Heroic	Invigorating / Joyous	Jubilant / Soulful	Ramshackle / Rollicking	Wild / Rowdy
	4	Happy / Soulful	Playful / Swingin'	Exuberant / Festive	Upbeat Pop Groove	Happy Excitement
	5	Party / Fun	Showy / Rousing	Lusty / Jaunty	Loud Celebratory	Euphoric Energy

## Navigating with Mood

This topic suggests some possible UI designs for mood-based playlists. The designs presented are suggestions only. The Mood APIs are flexible and can support most any type of UI that can be designed.

## Slider Navigation

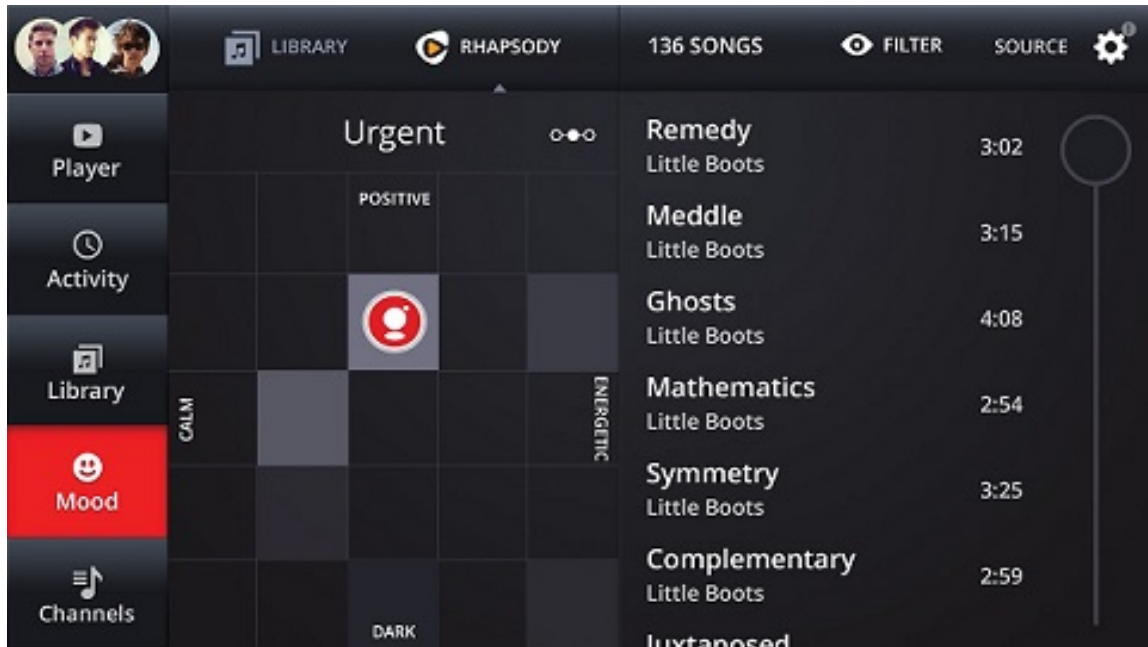
Slider Mood Navigation is shown below. Gracenote recommends this design because of its simplicity and ease-of-use in an automotive environment. It can support a touch-screen or scroll-wheel interface for selecting values. This design provides the ability to choose five discrete values on each dimension of Valence and Arousal.



## Grid Navigation

Grid navigation is shown below. It is similar to the slider design. However, mood selection is made across a two-dimensional grid of discrete mood values. With this design, the user can select 25 discrete values as cells on a two-dimensional grid of Valence/Arousal values. This design supports a touch-screen easily, but may be more difficult to map to a scroll-wheel interface. Grid navigation can implement a heat-map design to indicate the relative number of tracks matching each grid. For example, the more tracks there are in a cell, the darker the shade of

the cell. In this example, the vertical axis corresponds to Valence values, and the horizontal axis corresponds to Arousal values.



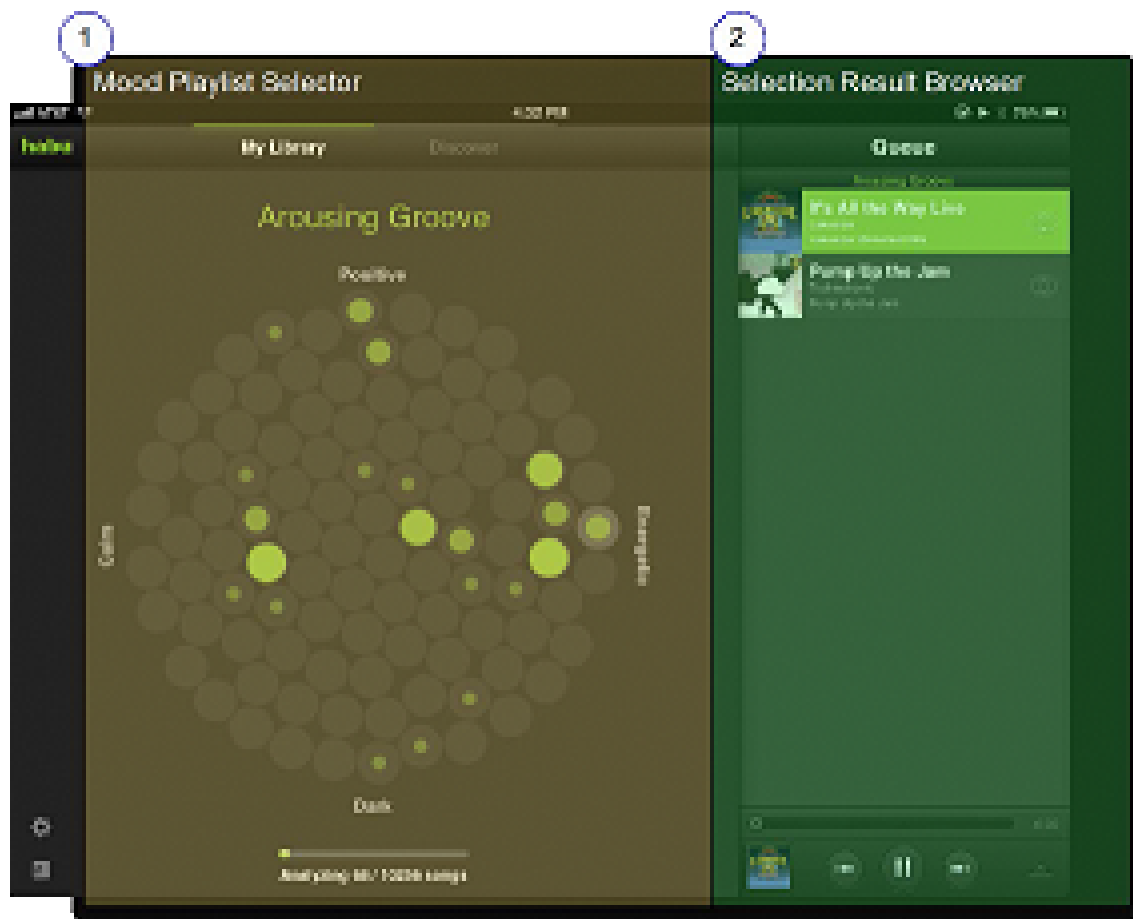
### ***Bubble Magnitude Navigation***

Bubble Magnitude navigation is shown below. This interface differs by using state bubbles to show the user what moods exist in the collection. Bubbles that are filled-in contain tracks for that mood. The size of the bubble indicates the relative number of tracks (the magnitude) in that mood category. This is similar to the heat-map design used in Grid navigation.

The bubbles are placed in a Valence/Arousal grid to provide an intuitive way to select mood playlists. The implementation supports a touch screen easily, but may be more difficult to map to a scroll-wheel interface..

The example below shows Bubble Magnitude navigation for the Level 2 set of 100 moods. This implementation is based on the Gracenote Habu mobile application for the iPad. Automotive applications should provide Level 1 moods

(25 bubbles). As in the other implementations, selecting a bubble creates a playlist for the Valence/Arousal value.



### ***Other Mood Design Considerations***

Whichever interface design you implement, your application should follow these guidelines when a user selects a mood.

- Keep the Mood Playlist Selector active so users can easily select different moods.

- When the user selects a new mood, go directly to the Selection Results Browser with the refreshed playlist.
- After several seconds of Mood being active, go directly to Selection Results Browser and start playback of the tracks. This allows the user to activate Mood and continue to use the mood selection from the last time Mood was used.

In addition to the UI designs presented above, you can add other components to enhance the mood experience. Examples of these enhancements are:

- **Voice Commands:** Mood playlist navigation by voice is a feature that can improve the automobile music listening experience. As a mood playlist is playing for instance, a user could speak More Positive or Less Dark to have the playlist automatically adjust to their tastes. All of the interfaces presented above could support a voice interface like this, but the Slider navigation interface visually maps more directly to these commands.
- **Genre Filtering:** The Grid navigation interface can be further enhanced to include a genre filter button. This filter lets the user to refine mood playlist to specific genres. To keep the filter selection simple, limit the available genres to only those in the user's collection. Your application can provide filters based on other track metadata as well, such as era and artist type.
- **100 Mood Option:** Both Grid and Bubble Magnitude navigation interfaces can support Level 2 (100 moods). The Slider navigation interface can also support Level 2 using two dimensions of 10 buttons each. However, the Level 1 (25 mood) design is optimal for this interface.

### Mood Level Arousal/Valance Values

The tables below list the Valence/Arousal value mappings for Level 1 and Level 2 mood levels.

#### ***Level 1 Mood Levels***

Mood	Valance (Emotion)	Arousal (Energy)
Somber	1	1



Mood	Valance (Emotion)	Arousal (Energy)
Melancholy	2	1
Sentimental	3	1
Tender	4	1
Peaceful	5	1
Gritty	1	2
Cool	2	2
Sophisticated	3	2
Romantic	4	2
Easygoing	5	2
Serious	1	3
Yearning	2	3
Sensual	3	3
Empowering	4	3
Upbeat	5	3
Brooding	1	4
Urgent	2	4
Fiery	3	4
Stirring	4	4
Lively	5	4
Aggressive	1	5
Defiant	2	5
Energizing	3	5
Rowdy	4	5
Excited	5	5

**Level 2 Mood Levels**

Mood	Valance (Emotion)	Arousal (Energy)
Dark Cosmic	1	1
Solemn / Spiritual	2	1
Wistful / Forlorn	3	1
Mysterious / Dreamy	4	1
Lyrical Sentimental	5	1
Tender / Sincere	6	1
Romantic / Lyrical	7	1
Refined / Mannered	8	1
Reverent / Healing	9	1
Pastoral / Serene	10	1
Creepy / Ominous	1	2
Enigmatic / Mysterious	2	2
Sad / Soulful	3	2
Light Melancholy	4	2
Cool Melancholy	5	2
Gentle Bittersweet	6	2
Light Groovy	7	2
Awakening / Stately	8	2
Quiet / Introspective	9	2
Delicate / Tranquil	10	2
Depressed / Lonely	1	3
Sober / Determined	2	3
Cool Confidence	3	3
Casual Groove	4	3

Mood	Valance (Emotion)	Arousal (Energy)
Intimate Bittersweet	5	3
Suave / Sultry	6	3
Dramatic / Romantic	7	3
Sweet / Sincere	8	3
Friendly	9	3
Hopeful / Breezy	10	3
Gritty / Soulful	1	4
Strumming Yearning	2	4
Dark Groovy	3	4
Wary / Defiant	4	4
Smoky / Romantic	5	4
Dark Playful	6	4
Lush / Romantic	7	4
Heartfelt Passion	8	4
Charming / Easygoing	9	4
Cheerful / Playful	10	4
Serious / Cerebral	1	5
Melodramatic	2	5
Sensitive / Exploring	3	5
Bittersweet Pop	4	5
Dreamy Pulse	5	5
Soft Soulful	6	5
Dramatic Emotion	7	5
Strong / Stable	8	5
Soulful / Easygoing	9	5
Carefree Pop	10	5

Mood	Valance (Emotion)	Arousal (Energy)
Thrilling	1	6
Hypnotic Rhythm	2	6
Energetic Dreamy	3	6
Energetic Yearning	4	6
Intimate	5	6
Sensual Groove	6	6
Idealistic / Stirring	7	6
Powerful / Heroic	8	6
Happy / Soulful	9	6
Party / Fun	10	6
Dreamy Brooding	1	7
Evocative / Intriguing	2	7
Dark Urgent	3	7
Dark Pop	4	7
Passionate Rhythm	5	7
Dark Sparkling Lyrical	6	7
Focused Sparkling	7	7
Invigorating / Joyous	8	7
Playful / Swingin	9	7
Showy / Rousing	10	7
Alienated / Brooding	1	8
Energetic Melancholy	2	8
Energetic Anxious	3	8
Dark Pop Intensity	4	8
Energetic Abstract Groove	5	8
Fiery Groove	6	8

Mood	Valance (Emotion)	Arousal (Energy)
Triumphant / Rousing	7	8
Jubilant / Soulful	8	8
Exuberant / Festive	9	8
Lusty / Jaunty	10	8
Chaotic / Intense	1	9
Dark Hard Beat	2	9
Attitude / Defiant	3	9
Heavy Brooding	4	9
Edgy / Sexy	5	9
Arousing Groove	6	9
Confident / Tough	7	9
Ramshackle / Rollicking	8	9
Upbeat Pop Groove	9	9
Loud Celebratory	10	9
Aggressive Power	1	10
Heavy Triumphant	2	10
Hard Dark Excitement	3	10
Hard Positive Excitement	4	10
Abstract Beat	5	10
Heavy Beat	6	10
Driving Dark Groove	7	10
Wild / Rowdy	8	10
Happy Excitement	9	10
Euphoric Energy	10	10

### **1.4.6** *Rhythm Overview*

Rhythm enables seamless integration of any audio source with online music services within a single application. Depending on the supported features of each online music service, an identified track can be used to link to a service to:

- Play that song
- Play more songs from that artist/album or
- Create an adaptive radio session

The identified track can be from any music source including terrestrial radio, CD, audio file or even another music service provided the track has been identified using Gracenote recognition technology.

Gracenote Rhythm allows you to create highly relevant and personalized adaptive radio stations and music recommendations for end users. Radio stations can be created with seed artists and tracks or a combination of genre, era, and mood, and can support Digital Millennium Copyright Act (DCMA) sequencing rules.

Rhythm includes the ability for an end user to specify whether they have played a track, like or dislike it, or want to skip past it. Rhythm can reconfigure the radio station playlist to more closely reflect end user preferences. In addition, it supports cover art retrieval, third-party links, and metadata content exploration.

Rhythm can:

- **Create a Radio Station:** Creating a radio station requires a User ID and a seed (artist, track, genre, era, or mood). Creating a radio station generates a track playlist.
- **Adapt to User Feedback:** End users can provide feedback (play, like, dislike, skip) about songs in the current radio station's playlist. This can result in modifying a playlist or generating a new one altogether.
- **Tune a Radio Station:** Reconfigure an existing radio station playlist based on track popularity and similarity.

- **Create a Playlist:** Rhythm can create a track playlist without the overhead of creating a radio station.

Radio station creation returns a radio station ID and a track playlist. The radio ID is used in subsequent actions on the station, its playlist, or its tuning parameters. Rhythm builds radio stations using GDOs.



**Note:** For object-oriented applications, for example, those written in C++, GDOs are the same thing as GnDataObjects. All object-oriented response objects are derived from GnDataObject.

MusicID can retrieve the necessary GDOs by looking up artist names, album titles, or track titles. To create a playlist, Rhythm uses:

- Artist ID
- Track TUI (Title Unique Identifier)
- Genre, mood, or era attributes
- More than one of the above in combination

Multiple seed stations create a greater variety of results. Once you create a radio station, you can examine its entire playlist (up to 25 tracks at a time). End users can take various feedback event actions on the playlist such as:

- Track like
- Track dislike
- Track skipped
- Track played
- Artist like
- Artist dislike

When feedback is sent, Rhythm reconfigures the station's playlist accordingly. Rhythm supports the retrieval of the playlist at any time before or after feedback is provided.

A station's behavior may also be tuned by changing playlist generation behavior. For example, you can get difference responses based on options for popularity or similarity.

### **1.4.7** *MusicID-Match Overview*

MusicID-Match (also known as Scan and Match) uses a combination of waveform fingerprinting technologies to match tracks within an end-users collection to tracks within a cloud music provider's catalog, enabling instant playback from all devices without requiring upload.

Full-file fingerprinting technology is employed to identify the differences between varying length versions of the same track, and a highly robust fingerprint allows for the differentiation of highly similar tracks, such as remasters and clean/explicit versions.

MusicID-Match supports:

- Queries using MusicID-Match fingerprint types
- Query results management

A MusicID-File fingerprint is first generated for the purposes of candidate generation. Metadata is utilized to resolve or further refine the track candidate set.

A full-file fingerprint, using fingerprinting technology similar to that of MusicID Stream, is generated to perform any final disambiguation. All fingerprint matching is performed against a private data store containing only the required reference fingerprints for the customer's catalog.

MusicID-Match requires that all matching logic be implemented service-side.

## **1.5** Image Formats and Dimensions

Gracenote provides images in several dimensions to support a variety of applications. Applications or devices must specify image size when requesting an



image from Gracenote. All Gracenote images are provided in the JPEG (.jpg) image format.

### **1.5.1 Available Image Dimensions**

Gracenote provides images to fit within the following six square dimensions. All image sizes are available online and the most common 170x170 and 300x300 sizes are available in a local database.

Image Dimension Name	Pixel Dimensions
75	75 x 75
170	170 x 170
300	300 x 300
450	450 x 450
720	720 x 720
1080	1080 x 1080

Please contact your Gracenote representative for more information.

Source images are not always square, and may be proportionally resized to fit within the specified square dimensions. Images will always retain their original aspect ratio.

### **1.5.2 Common Media Image Dimensions**

Media images exist in a variety of dimensions and orientations. Gracenote resizes ingested images according to carefully developed guidelines to accommodate these image differences, while still optimizing for both developer integration and the end-user experience.

#### **1.5.2.1 Music Cover Art**

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

### 1.5.2.2 Artist Images

Artist and contributor images, such as publicity photos, come in a wide range of sizes and both portrait and landscape orientations.

Video contributor images are most often provided in portrait orientation.

### 1.5.2.3 Genre Images

Genre Images are provided by Gracenote to augment Cover Art and Artist Images when unavailable and to enhance the genre navigation experience. They are square photographic images and cover most of the Level 1 hierarchy items.

### 1.5.2.4 Video Cover Art

Video cover art is most often taller than it is wide (portrait orientation). For most video cover art, this means that images will completely fill the vertical dimension of the requested image size, and will not fill the horizontal dimension. Therefore, while mostly fixed in height, video images may vary slightly in width. For example, requests for a "450" video image will likely return an image that is exactly 450 pixels tall, but close to 300 pixels wide.

As with CD cover art, the dimensions of video covers also include packaging variants such as box sets which sometimes result in significant variations in video image dimensions.

#### Video Image Dimension Variations

Video imagery commonly conforms to the shape of a tall rectangle with either a 3:4 or 6:9 aspect ratio. Image dimension characteristics of Gracenote Video imagery are provided in the following sections as guidelines for customers who want to implement Gracenote imagery into UI designs that rely on these aspect ratios. Gracenote recommends, however, that applications reserve square spaces in UI designs to accommodate natural variations in image dimensions.

### ***Video (AV Work) Images***

AV Work images typically conform to a 3:4 (width:height) aspect ratio.

3:4 (± 10%)	Narrower	Wider	Narrowest	Widest
98%	1%	1%	1:2	9:5

### ***Video Product Images***

Video Product images typically conform to a 3:4 (width:height) aspect ratio

3:4 (± 10%)	Narrower	Wider	Narrowest	Widest
90%	5%	5%	1:3	5:1

### ***Video Contributor Images***

Video Contributor images typically conform to a 6:9 (width:height) aspect ratio. Two ranges are provided due to larger variation in image dimensions.

6:9 (± 20%)	Narrower	Wider	Narrowest	Widest
90%	0%	10%	1:3	9:5

6:9 (± 10%)	Narrower	Wider	Narrowest	Widest
69%	1%	30%	1:3	9:5

## ***1.5.3 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.

### 1.5.3.1 Using a Default Image When Cover Art Is Missing

Occasionally, an Album result might have missing cover art. If the cover art is missing, Gracenote recommends trying to retrieve the artist image, and if that is not available, trying to retrieve the genre image. If none of these images are available, your application can use a default image as a substitute. Gracenote distributes an clef symbol image to serve as a default replacement. The images are in jpeg format and located in the /images folder of the package. The image names are:

- music\_75x75.jpg
- music\_170x170.jpg
- music\_300x300.jpg

### 1.5.3.2 Image Resizing Guidelines

Gracenote images are designed to fit 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.

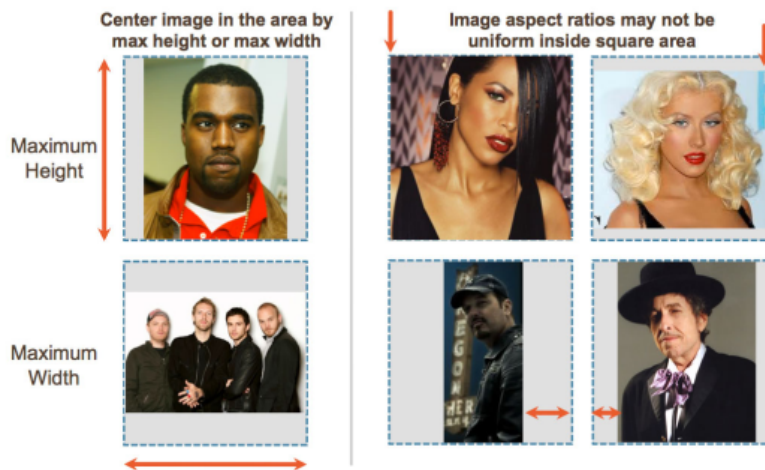
Gracenote resizes images based on the following guidelines:

- **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.



**Note:** 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.

## Chapter 2 Setup and Samples

This section describes how to setup your development environment to create GNSDK applications. It also describes how to use sample applications to get you started with development.

### 2.1 Modules Overview

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

The table below describes the primary modules provided by the GNSDK. The actual modules in your software package depends on your product and as specified in your Gracenote license agreement. The modules are listed alphabetically.

Module	Description
ACR module	Supports Automatic Content Recognition (ACR).
EPG module	Supports Electronic Programming Guide (EPG) features
FP Local module	Provides the option to store fingerprint data in local cache for faster local lookup.
Link module	A deprecated module that provides links to enriched metadata. Instead use the fetch APIs in Manager.
Manager module	Main module used by all applications. Other GNSDK libraries cannot function without it.

MoodGrid module	Provides easy-to-use, spatially-aware music organization features to manage a user's music collection based on a set of Gracenote attributes.
DSP (Music Fingerprinting) module	Provides digital signal processing functionality used by other GNSDK libraries.
Music Lookup Local module	Provides services for local lookup of various identification queries such as text and CD TOC search. Also provides services for local retrieval of content such as cover art. When enabled other GNSDK query objects are able to perform local lookups for various types of Gracenote searches.
Music Lookup Local Stream module	Provides services for local lookup of MusicID Stream queries. The MusicID Stream local database is constructed from "bundles" provided periodically by Gracenote. Your application must ingest the bundle, a process that adds the tracks from the bundle to the local database making them available for local recognition.
MusicID module	Provides for audio recognition using CD TOC-based search (MusicID CD), Text-based searches (MusicID Text), Fingerprint, and Identifier lookup functionality.
MusicID-File module	Provides a more comprehensive music file identification process than MusicID. MusicID-File is intended to be used with collections of file-based media.
MusicID Stream	Provides software to recognize music delivered as a continuous stream. It automatically manages buffering of streaming audio, and continuously fingerprints the audio stream. It does not retain these fingerprints for later re-use.
Playlist module	Supports creating lists of related songs from a music collection. It also supports "More Like This" playlists to generate optimal playlist results and eliminates the need to create and validate Playlist Definition Language statements.

Rhythm module	Supports creation of personalized adaptive radio stations and music recommendations for end users. Use seed artists and tracks, or a combination of genre, era, and mood to create the stations.
SQLite module	Provides a local storage solution for GNSDK using the SQLite database engine. This module is primarily used to manage a local cache of queries and content that the GNSDK libraries make to the Gracenote Service.
Submit module	Provides functionality necessary to submit metadata parcels to the Gracenote database.

### **2.1.1** *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.

#### **2.1.1.1** GNSDK Modules

- DSP
- LINK
- LOOKUP\_LOCAL
- LOOKUP\_LOCALSTREAM
- MANAGER
- MOOD
- MUSICID
- MUSICID\_FILE
- MUSICID\_MATCH
- MUSICID\_STREAM
- PLAYLIST



- STORAGE\_SQLITE
- SUBMIT

## 2.2 Using the Sample Applications

GNSDK provides working command-line sample applications that demonstrate common queries and application scenarios.

The package also provides sample databases you can use when developing your applications.

Gracenote recommends stepping through the sample applications with a debugger to observe module usage and API calls.

### 2.2.1 Sample Applications and Reference Applications

Below is a list of sample applications and reference applications for GNSDK. Sample and reference applications are included in the release package in the /samples and /reference\_applications folders.

- **Sample applications** demonstrate common queries to identify media elements and access metadata.
- **Reference applications** are more extensive and demonstrate specialized or more comprehensive use cases.

Gracenote recommends that you use the sample and reference applications as a basis for any applications you develop.

#### 2.2.1.1 Common Flow and Layout

The sample applications are designed to have the same look and feel, and the same layout and flow, so developers can find their way around them quickly once they become familiar with one of them. In theory, you could do a diff between two samples and only see the meaningful differences - essential calls, and so on, for a given feature. For this reason, they contain some or all of the following common set of utility functions:

- **\_init\_gnsdk()** - Validate the caller, initialize the SDK Manager and modules, get a user handle, enable logging, initialize local storage and cache, and load a locale. Note that not all these initializations are done in every program.
- **\_get\_user\_handle()** - Load an existing user handle from file storage, or register a new one.
- **\_enable\_logging()** - Enable SDK logging. This helps Gracenote debug your application, if necessary.
- **\_display\_gnsdk\_product\_info()** - Display product version information.
- **\_display\_last\_error()** - Echo system information from last error.
- **\_set\_locale()** - Set application locale. Note that this is only necessary if you are using locale-dependant fields such as genre, mood, origin, era, and so on. Your app may or may not be accessing locale\_dependent fields, but it does not hurt to do this initialization as a matter of course.
- **\_shutdown\_gnsdk()** - Calls shutdown on all initialized GNSDK modules. If the user handle has changed during program execution, it is saved out to file storage.
- **\_display\_embedded\_db\_info()** - Display local Gracenote database information.

### 2.2.1.2 Sample Applications

Name (Click to view in documentation)	Modules Used	Description	Location in Package
moodgrid	Playlist	Provides a complete Mood sample application.	samples/moodgrid/main.c
musicid_file_albumid	MusicID-File	Uses AlbumID for advanced audio recognition. The audio file's folder location and its similarity to other audio files are used to achieve more accurate identification.	samples/musicid_file_albumid/main.c

Name (Click to view in documentation)	Modules Used	Description	Location in Package
<a href="#">musicid_file_libraryid</a>	MusicID-File	Uses LibraryID to perform additional scanning and processing of all the files in an entire collection. This enables LibraryID to find groupings that are not captured by AlbumID processing	<code>samples/musicid_file_libraryid/main.c</code>
<a href="#">musicid_file_trackid</a>	MusicID-File	Uses TrackID, the simplest MusicID-File processing method to process each audio file independently, without regard for any other audio files in a collection.	<code>samples/musicid_file_trackid/main.c</code>
<a href="#">musicid_gdo_navigation</a>	MusicID	Uses MusicID to look up Album GDO content, including Album artist, credits, title, year, and genre. It demonstrates how to navigate the album GDO that returns basic track information, including artist, credits, title, track number, and genre.	<code>samples/musicid_gdo_navigation/main.c</code>
<a href="#">musicid_image_fetch</a>	Link	Does a text search and finds images based on gdo type (album or contributor). It also finds an image based on genre.	<code>samples/musicid_image_fetch/main.c</code>
<a href="#">musicid_lookup_album_id</a>	MusicID	Looks up an album based on its TOC.	<code>samples/musicid_lookup_album_id/main.c</code>
<a href="#">musicid_lookup_album_local_online</a>	MusicID	Looks up an Album using a TUI in the local database, if not found then performs an online lookup.	<code>samples/musicid_lookup_album_local_online/main.c</code>

Name (Click to view in documentation)	Modules Used	Description	Location in Package
musicid_lookup_album_toc	MusicID	Looks up an Album based on its TOC using either a local database or online.	samples/musicid_lookup_album_toc/main.c
musicid_lookup_matches_text	MusicID	Performs a sample text query with album, track and artist inputs.	samples/musicid_lookup_matches_text/main.c
musicid_stream_manual	MusicID Stream	Uses MusicID Stream to fingerprint and identify a music track manually.	samples/musicid_stream_manual/main.c
playlist	Playlist	Demonstrates using the Playlist APIs to create More Like This and custom playlists.	samples/playlist/main.c
rhythm	Rhythm	Demonstrates the Rhythm SDK.	samples/rhythm/main.c
submit_album	Submit	Demonstrates editing an Album GDO and submitting it to Gracenote	samples/submit_album/main.c
submit_feature	Submit	Demonstrates processing music features and submitting a parcel to Gracenote	samples/submit_feature/main.c

### 2.2.1.3 Reference Applications

Name	Modules Used	Description	Location in Package
MoodGrid_MoreLikeThis	Playlist	Demonstrates the MoreLikeThis use-case using Playlist and Mood.	reference_apps/MoodGrid_MoreLikeThis

## 2.2.2 GNSDK C Quick Start Tutorial

### 2.2.2.1 Introduction

This tutorial shows you how to build a simple GNSDK C application from scratch. The application does a track title text search to identify an album in the Gracenote database. The Gracenote SDK provides several methods to access Gracenote metadata from a C application, most of which are demonstrated in the C sample apps that come with GNSDK.

**NOTE:** This tutorial application creates a very basic GNSDK application only, and is not meant to be used in a production environment. It does not include many other features and best practices found within the GNSDK and other sample applications. These include:

- GNSDK logging
- Loading a locale - Locales are a convenient way to group locale-dependent metadata. The quick start application does not access any locale-dependent metadata here.
- Code for accessing an embedded database
- Code for the SQLite module - Used for caching
- Decision code - Sometimes more than one match is returned or Gracenote is not entirely confident about the one match. This requires the user to pick one match or confirm the match. This application simply uses the first match.
- Partial or full metadata code - A match can contain full or partial metadata. If partial, an additional call is needed to get full metadata. In this case, the album title is part of partial metadata, so there is no need for an additional check.
- SDK user object - Normally, this object should be saved and retrieved from disk after initial creation. Here, the application just creates a new one.

Refer to the sample applications for examples of using these features in your application.

### 2.2.2.2 Setting Up Your Development Environment

The GNSDK is supports several different systems, including different flavors of Android, iOS, Linux, Macintosh, and Windows. Included with the SDK are the support files necessary to create a local build environment of your choosing.

#### Include Files

Include files for GNSDK applications are located in `/include` under your installation directory. There are approximately 30 platform independent include files, and one platform dependent file, `gnsdk_platform.h`. The different versions of the platform dependent file are stored under the `/include` directory in subdirectories named for specific supported platforms, for example, `linux_arm-32`. For proper compilation, ensure that the correct include files are made available to your build system. Although we mention these files for clarity, you should not need to include anything other than `gnsdk.h` in your application for GNSDK. It will include the files necessary for compilation, based on the options you have defined.

#### Library Files

The library files necessary for linking are placed into two directories. The directory `/lib` is for dynamically linked libraries, and `/lib-static` is for statically linked libraries. There are approximately 15 (Windows has double, including DLLs) library files containing the binaries for GNSDK. For your application's link step to locate and include the GNSDK functionality, you must make these libraries available to your link process.

#### Source Files

The SDK comes with several C samples. They are located in the `/samples` directory, with each sample having its own subdirectory. Most of these subdirectories consist of a `main.c` file and a GNU makefile. You can use a sample makefile as a guide to creating a makefile for this tutorial. You can either

add GNU software to your build system, such as with Cygwin on Windows, or you can build applications through native tools, such as Visual Studio.

### Executing The Application

All the sample make files create a `sample.exe` executable. For the quick start tutorial application, once you build successfully, you should be able to run the executable with the following command at a command prompt:

```
> sample <clientID> <clientIDtag> licfile.txt
```

In this example, the license file text is in its own file as a single line of text. You get Client ID and license information from Gracenote.

#### 2.2.2.3 Coding the App

Once your development environment is setup, you can begin coding the application. As in the other sample applications, the code is going to be in one `main.c` file which you can create in any text editor or IDE. The application performs the following steps:

1. Perform GNSDK initializations
2. Look up an album in Gracenote's database from a single track
3. Display the album title of the first match
4. Shut down the GNSDK and exit the program

### Code the Includes, Defines and Local Function Declarations

First, code all your `pre-main()` function includes, defines and local function declarations:

```
#define GNSDK_MUSICID      1
#include "gnsdk.h" // Includes all GN SDK headers
/* Standard C headers - used by the sample app, but not required for
GNSDK */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
/* Local function declarations */
static int _init_gnsdk(const char* client_id, const char* client_id_
tag, const char* client_app_version,
                    const char* license_path, gnsdk_user_handle_t* p_user_
handle);
static void _display_error(int line_num, const char* info, gnsdk_
error_t error_code);
static void _shutdown_gnsdk(gnsdk_user_handle_t user_handle);
static void _query_album_lookup(gnsdk_user_handle_t user_handle);
static void _display_album_gdo(gnsdk_gdo_handle_t album_gdo);
```

A define is required for each module (MusicID, MusicID-File, DSP, and so on.) that the program uses. This application only uses MusicID. Including "gnsdk.h" will automatically include all the GNSDK headers. We are also including forward declarations for all our local functions except `main()`.

### Code the `main()` Function

The `main()` function validates the command-line parameters and calls three other functions to perform initialization, lookup and display, and shutdown:

```
int main(int argc, char* argv[])
{
    gnsdk_user_handle_t user_handle      = GNSDK_NULL;
    const char*         client_id        = NULL;
    const char*         client_id_tag    = NULL;
    const char*         client_app_version = "1";    // Version of your
application
    const char*         license_path     = NULL;
    if (argc != 4)
    {
        printf("\nUsage:\n%s clientid clientidtag license\n", argv[0]);
        return -1;
    }
    client_id      = argv[1];
    client_id_tag  = argv[2];
    license_path   = argv[3];
    /* GNSDK initialization */
    if ((_init_gnsdk(client_id, client_id_tag, client_app_
version, license_path, &user_handle)) == 0)
    {
        _query_album_lookup(user_handle);    /* Perform a sample album
```



```
text search query - calls display function */
    _shutdown_gnsdk(user_handle);    /* Clean up and shutdown */
}
return 0;
}
```

## Code GNSDK Initializations

The `_init_gnsdk()` function performs all necessary GNSDK initializations which include the following:

- Initialize the GNSDK manager with your license text
- Initialize the MusicID library
- Initialize the SDK User object - Normally, you would save this to a file after creating it and save it again at program exit if it had changed. This tutorial, skips storage and just creates a new user object and releases it on exit without saving it. User ID's are consumable. Best practice is to create and save it, rather than discarding it.

All the GNSDK functions return "GNSDK\_SUCCESS" if completed successfully. The error handling function - `_display_error()` grabs system error information and displays it to the console when an error occurs.

```
static void _display_error(int line_num, const char* info, gnsdk_
error_t error_code)
{
    const gnsdk_error_info_t* error_info = gnsdk_manager_error_info();
    printf("\nerror 0x%08x - %s\n\tline %d, info %s\n", error_code,
error_info->error_description, line_num, info);
}

static int _init_gnsdk(const char* client_id, const char* client_id_
tag, const char* client_app_version, const char* license_path,
gnsdk_user_handle_t* p_user_handle)
{
    gnsdk_manager_handle_t sdkmgr_handle = GNSDK_NULL;
    gnsdk_error_t error = GNSDK_SUCCESS;
    gnsdk_user_handle_t user_handle = GNSDK_NULL;
    gnsdk_str_t serialized_user = GNSDK_NULL;
```

```
/* Initialize the GNSDK Manager */
if ((error = gnsdk_manager_initialize(&sdmgrp_handle, license_path,
GNSDK_MANAGER_LICENSEDATA_FILENAME)) != GNSDK_SUCCESS)
{
    _display_error(__LINE__, "gnsdk_manager_initialize()", error);
}

/* Initialize the MusicID Library */
else if ((error = gnsdk_musicid_initialize(sdkmgr_handle)) != GNSDK_
SUCCESS)
{
    _display_error(__LINE__, "gnsdk_musicid_initialize()", error);
}
else if (( error = gnsdk_manager_user_register(GNSDK_USER_REGISTER_
MODE_ONLINE, client_id, client_id_tag, client_app_version,
&serialized_user)) != GNSDK_SUCCESS)
{
    _display_error(__LINE__, "gnsdk_manager_user_register()", error);
}
else if (( error = gnsdk_manager_user_create(serialized_user,
client_id, &user_handle)) != GNSDK_SUCCESS)
{
    _display_error(__LINE__, "gnsdk_manager_user_create()", error);
}

if (error != GNSDK_SUCCESS)
    _shutdown_gnsdk(user_handle);
else *p_user_handle = user_handle;

return error;
}
```

## Code the Lookup Function

The `_query_album_lookup` function looks up an album using a track title ("I Wonder"), which returns "Graduation" by Kanye West. This function uses what are called "GDOs" (Gracenote Data Objects) that receive both metadata and queries. GDOs are an important part of the GNSDK. To find out more about them, see [About Gracenote Data Objects \(GDOs\)](#).

```
static void _query_album_lookup( gnsdk_user_handle_t user_handle)
{
```

```
gnsdk_error_t      error      = GNSDK_SUCCESS;
gnsdk_musicid_query_handle_t query_handle = GNSDK_NULL;
gnsdk_gdo_handle_t response_gdo = GNSDK_NULL;
gnsdk_gdo_handle_t album_gdo   = GNSDK_NULL;
gnsdk_uint32_t     count      = 0;
printf("\nMusicID Text Search Query\n");
/* Create the query handle */
if ((error = gnsdk_musicid_query_create(user_handle, GNSDK_NULL,
GNSDK_NULL, &query_handle)) == GNSDK_SUCCESS)
{
    if ((error = gnsdk_musicid_query_set_text( query_handle, GNSDK_
MUSICID_FIELD_TITLE, "I Wonder")) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_musicid_query_set_text()",
error);
    }
    else if ((error = gnsdk_musicid_query_find_albums(query_handle,
&response_gdo)) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_musicid_query_find_albums()",
error);
    }
    /* Find number of matches */
    else if ((error = gnsdk_manager_gdo_child_count(response_gdo, GNSDK_
GDO_CHILD_ALBUM, &count)) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_gdo_child_count(GNSDK_GDO_
CHILD_ALBUM)", error);
    }
    else if (count == 0)
    {
        printf("\nNo albums found for the input.\n");
    }
    /* Get first child Album GDO */
    else if ((error = gnsdk_manager_gdo_child_get(response_gdo, GNSDK_
GDO_CHILD_ALBUM, 1, &album_gdo)) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_gdo_child_get(GNSDK_GDO_
CHILD_ALBUM)", error);
    }
    else
    {
        _display_album_gdo(album_gdo);
        gnsdk_manager_gdo_release(album_gdo);
        album_gdo = GNSDK_NULL;
    }
}
```

```
    }  
    }  
    /* Clean up - release the query handle and results */  
    if (GNSDK_NULL != query_handle) gnsdk_musicid_query_release(query_  
handle);  
    if (GNSDK_NULL != response_gdo) gnsdk_manager_gdo_release(response_  
gdo);  
}
```

## Code the Display Function

This function parses the album title from our returned GDO structure and displays it to the console.

```
static void _display_album_gdo(gnsdk_gdo_handle_t album_gdo)  
{  
    gnsdk_error_t    error    = GNSDK_SUCCESS;  
    gnsdk_gdo_handle_t title_gdo = GNSDK_NULL;  
    gnsdk_cstr_t     value    = GNSDK_NULL;  
    /* Album Title */  
    if ((error = gnsdk_manager_gdo_child_get( album_gdo, GNSDK_GDO_  
CHILD_TITLE_OFFICIAL, 1, &title_gdo )) == GNSDK_SUCCESS)  
    {  
        if ((error = gnsdk_manager_gdo_value_get( title_gdo, GNSDK_GDO_  
VALUE_DISPLAY, 1, &value )) == GNSDK_SUCCESS)  
        {  
            printf( "%8s %s\n", "Album Title:", value );  
        }  
        else _display_error(__LINE__, "gnsdk_manager_gdo_value_get()",  
error);  
        gnsdk_manager_gdo_release(title_gdo);  
    }  
    else _display_error(__LINE__, "gnsdk_manager_gdo_child_get()",  
error);  
}
```

## Code the Shutdown Function

This function releases the user object handle and shuts down GNSDK libraries

```
static void _shutdown_gnsdk(gnsdk_user_handle_t user_handle)
```

```
{
    gnsdk_error_t error          = GNSDK_SUCCESS;
    /* Release our user handle */
    if ((error = gnsdk_manager_user_release(user_handle)) != GNSDK_
SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_user_release()", error);
    }
    /* Shutdown the libraries */
    gnsdk_manager_shutdown(); // you only need to call this.  It will
shut down all open libraries.
}
```

#### 2.2.2.4 Conclusion

Once you successfully build and run the app, you should see the following output similar to the following:

```
MusicID Text Search Query
Album Title: Graduation
```

This quick start sample demonstrates only a very small portion of the GNSDK capabilities. Check out the full documentation and the samples included with GNSDK to see how to search for different types of metadata by audio fingerprint, text, or existing file.

#### 2.2.2.5 Reference

The complete `main.c` file should look like this:

```
#define GNSDK_MUSICID          1

#include "gnsdk.h" // Includes all GN SDK headers

/* Standard C headers - used by the sample app, but not required for
GNSDK */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* Local function declarations */
static int _init_gnsdk(const char* client_id, const char* client_id_
```

```
tag, const char* client_app_version,
        const char* license_path, gnsdk_user_handle_t* p_user_
handle);
static void _display_error(int line_num, const char* info, gnsdk_
error_t error_code);
static void _shutdown_gnsdk(gnsdk_user_handle_t user_handle);
static void _query_album_lookup(gnsdk_user_handle_t user_handle);
static void _display_album_gdo(gnsdk_gdo_handle_t album_gdo);

int main(int argc, char* argv[])
{
    gnsdk_user_handle_t user_handle      = GNSDK_NULL;
    const char*        client_id         = NULL;
    const char*        client_id_tag     = NULL;
    const char*        client_app_version = "1";
    const char*        license_path      = NULL;

    int                rc                = 0;

    if (argc != 4)
    {
        printf("\nUsage:\n%s clientid clientidtag license\n", argv[0]);
        return -1;
    }

    client_id   = argv[1];
    client_id_tag = argv[2];
    license_path = argv[3];

    /* GNSDK initialization */
    if ((rc = _init_gnsdk(client_id, client_id_tag, client_app_
version, license_path, &user_handle)) == 0)
    {
        _query_album_lookup(user_handle); /* Perform a sample album TOC
query */
        _shutdown_gnsdk(user_handle);    /* Clean up and shutdown */
    }

    return rc;
}

static void _display_error(int line_num, const char* info, gnsdk_
error_t error_code)
```

```
{
    const gnsdk_error_info_t* error_info = gnsdk_manager_error_info();
    printf("\nerror 0x%08x - %s\n\tline %d, info %s\n", error_code,
error_info->error_description, line_num, info);
}

static int _init_gnsdk(const char* client_id, const char* client_id_
tag, const char* client_app_version, const char* license_path,
                    gnsdk_user_handle_t* p_user_handle)
{
    gnsdk_manager_handle_t sdkmgr_handle = GNSDK_NULL;
    gnsdk_error_t          error          = GNSDK_SUCCESS;
    gnsdk_user_handle_t    user_handle    = GNSDK_NULL;
    gnsdk_str_t            serialized_user = GNSDK_NULL;

    /* Initialize the GNSDK Manager */
    if ((error = gnsdk_manager_initialize(&sdkmgr_handle, license_path,
GNSDK_MANAGER_LICENSEDATA_FILENAME)) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_initialize()", error);
    }

    /* Initialize the MusicID Library */
    else if ((error = gnsdk_musicid_initialize(sdkmgr_handle)) != GNSDK_
SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_musicid_initialize()", error);
    }
    else if ((error = gnsdk_manager_user_register(GNSDK_USER_REGISTER_
MODE_ONLINE, client_id, client_id_tag, client_app_version,
&serialized_user)) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_user_register()", error);
    }
    else if ((error = gnsdk_manager_user_create(serialized_user,
client_id, &user_handle)) != GNSDK_SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_user_create()", error);
    }

    if (error != GNSDK_SUCCESS)
        _shutdown_gnsdk(user_handle);
}
```

```
    else *p_user_handle = user_handle;

    return error;
}

static void _shutdown_gnsdk(gnsdk_user_handle_t user_handle)
{
    gnsdk_error_t error = GNSDK_SUCCESS;

    /* Release our user handle - note that if updated_serialized_user_
    string is not null, meaning */
    /* the handle had changed during program execution, then you should
    save it out to disk again */
    if ((error = gnsdk_manager_user_release(user_handle)) != GNSDK_
    SUCCESS)
    {
        _display_error(__LINE__, "gnsdk_manager_user_release()", error);
    }

    /* Shutdown the libraries */
    gnsdk_manager_shutdown();
}

static void _query_album_lookup( gnsdk_user_handle_t user_handle)
{
    gnsdk_error_t error = GNSDK_SUCCESS;
    gnsdk_musicid_query_handle_t query_handle = GNSDK_NULL;
    gnsdk_gdo_handle_t response_gdo = GNSDK_NULL;
    gnsdk_gdo_handle_t album_gdo = GNSDK_NULL;
    gnsdk_uint32_t count = 0;

    printf("\nMusicID Text Search Query\n");

    /* Create the query handle */
    if ((error = gnsdk_musicid_query_create(user_handle, GNSDK_NULL,
    GNSDK_NULL, &query_handle)) == GNSDK_SUCCESS)
    {
        if ((error = gnsdk_musicid_query_set_text( query_handle, GNSDK_
    MUSICID_FIELD_TITLE, "I Wonder")) != GNSDK_SUCCESS)
        {
            _display_error(__LINE__, "gnsdk_musicid_query_set_text()",
            error);
        }
    }
}
```



```
        else if ((error = gnsdk_musicid_query_find_albums(query_handle,
&response_gdo)) != GNSDK_SUCCESS)
        {
            _display_error(__LINE__, "gnsdk_musicid_query_find_albums()",
error);
        }
        /* Find number of matches */
        else if ((error = gnsdk_manager_gdo_child_count(response_gdo,
GNSDK_GDO_CHILD_ALBUM, &count)) != GNSDK_SUCCESS)
        {
            _display_error(__LINE__, "gnsdk_manager_gdo_child_count(GNSDK_
GDO_CHILD_ALBUM)", error);
        }
        else if (count == 0)
        {
            printf("\nNo albums found for the input.\n");
        }

        /* Get first child Album GDO */
        else if ((error = gnsdk_manager_gdo_child_get(response_gdo, GNSDK_
GDO_CHILD_ALBUM, 1, &album_gdo)) != GNSDK_SUCCESS)
        {
            _display_error(__LINE__, "gnsdk_manager_gdo_child_get(GNSDK_GDO_
CHILD_ALBUM)", error);
        }
        else
        {
            _display_album_gdo(album_gdo);
            gnsdk_manager_gdo_release(album_gdo);
            album_gdo = GNSDK_NULL;
        }
    }
    else
    {
        _display_error(__LINE__, "gnsdk_musicid_query_create", error);
    }

    /* Clean up - release the query handle and results */
    if (GNSDK_NULL != query_handle) gnsdk_musicid_query_release(query_
handle);
    if (GNSDK_NULL != response_gdo) gnsdk_manager_gdo_release(response_
gdo);
}
```

```
static void _display_album_gdo(gnsdk_gdo_handle_t album_gdo)
{
    gnsdk_error_t      error      = GNSDK_SUCCESS;
    gnsdk_gdo_handle_t title_gdo  = GNSDK_NULL;
    gnsdk_cstr_t       value      = GNSDK_NULL;

    printf("\nIn display_album_gdo\n");
    /* Album Title */
    if ((error = gnsdk_manager_gdo_child_get( album_gdo, GNSDK_GDO_
CHILD_TITLE_OFFICIAL, 1, &title_gdo )) == GNSDK_SUCCESS)
    {
        if ((error = gnsdk_manager_gdo_value_get( title_gdo, GNSDK_GDO_
VALUE_DISPLAY, 1, &value )) == GNSDK_SUCCESS)
        {
            printf( "%8s %s\n", "Album Title:", value );
        }
        else _display_error(__LINE__, "gnsdk_manager_gdo_value_get()",
error);

        gnsdk_manager_gdo_release(title_gdo);
    }
    else _display_error(__LINE__, "gnsdk_manager_gdo_child_get()",
error);
}
```

### **2.2.3** *Using the Sample Applications*

GNSDK provides working command-line sample applications that demonstrate common queries and application scenarios.

The package also provides sample databases you can use when developing your applications.

Gracenote recommends stepping through the sample applications with a debugger to observe module usage and API calls.

#### **2.2.3.1** *Sample Applications and Reference Applications*

Below is a list of sample applications and reference applications for GNSDK. Sample and reference applications are included in the release package in the

/samples and /reference\_applications folders.

- **Sample applications** demonstrate common queries to identify media elements and access metadata.
- **Reference applications** are more extensive and demonstrate specialized or more comprehensive use cases.

Gracenote recommends that you use the sample and reference applications as a basis for any applications you develop.

### Common Flow and Layout

The sample applications are designed to have the same look and feel, and the same layout and flow, so developers can find their way around them quickly once they become familiar with one of them. In theory, you could do a diff between two samples and only see the meaningful differences - essential calls, and so on, for a given feature. For this reason, they contain some or all of the following common set of utility functions:

- **\_init\_gnsdk()** - Validate the caller, initialize the SDK Manager and modules, get a user handle, enable logging, initialize local storage and cache, and load a locale. Note that not all these initializations are done in every program.
- **\_get\_user\_handle()** - Load an existing user handle from file storage, or register a new one.
- **\_enable\_logging()** - Enable SDK logging. This helps Gracenote debug your application, if necessary.
- **\_display\_gnsdk\_product\_info()** - Display product version information.
- **\_display\_last\_error()** - Echo system information from last error.
- **\_set\_locale()** - Set application locale. Note that this is only necessary if you are using locale-dependant fields such as genre, mood, origin, era, and so on. Your app may or may not be accessing locale\_dependent fields, but it does not hurt to do this initialization as a matter of course.

- **\_shutdown\_gnsdk()** - Calls shutdown on all initialized GNSDK modules. If the user handle has changed during program execution, it is saved out to file storage.
- **\_display\_embedded\_db\_info()** - Display local Gracenote database information.

## Sample Applications

Name (Click to view in documentation)	Modules Used	Description	Location in Package
moodgrid	Playlist	Provides a complete Mood sample application.	samples/moodgrid/main.c
musicid_file_albumid	MusicID-File	Uses AlbumID for advanced audio recognition. The audio file's folder location and its similarity to other audio files are used to achieve more accurate identification.	samples/musicid_file_albumid/main.c
musicid_file_libraryid	MusicID-File	Uses LibraryID to perform additional scanning and processing of all the files in an entire collection. This enables LibraryID to find groupings that are not captured by AlbumID processing	samples/musicid_file_libraryid/main.c
musicid_file_trackid	MusicID-File	Uses TrackID, the simplest MusicID-File processing method to process each audio file independently, without regard for any other audio files in a collection.	samples/musicid_file_trackid/main.c

Name (Click to view in documentation)	Modules Used	Description	Location in Package
musicid_gdo_navigation	MusicID	Uses MusicID to look up Album GDO content, including Album artist, credits, title, year, and genre. It demonstrates how to navigate the album GDO that returns basic track information, including artist, credits, title, track number, and genre.	samples/musicid_gdo_navigation/main.c
musicid_image_fetch	Link	Does a text search and finds images based on gdo type (album or contributor). It also finds an image based on genre.	samples/musicid_image_fetch/main.c
musicid_lookup_album_id	MusicID	Looks up an album based on its TOC.	samples/musicid_lookup_album_id/main.c
musicid_lookup_album_local_online	MusicID	Looks up an Album using a TUI in the local database, if not found then performs an online lookup.	samples/musicid_lookup_album_local_online/main.c
musicid_lookup_album_toc	MusicID	Looks up an Album based on its TOC using either a local database or online.	samples/musicid_lookup_album_toc/main.c
musicid_lookup_matches_text	MusicID	Performs a sample text query with album, track and artist inputs.	samples/musicid_lookup_matches_text/main.c
musicid_stream_manual	MusicID Stream	Uses MusicID Stream to fingerprint and identify a music track manually.	samples/musicid_stream_manual/main.c
playlist	Playlist	Demonstrates using the Playlist APIs to create More Like This and custom playlists.	samples/playlist/main.c
rhythm	Rhythm	Demonstrates the Rhythm SDK.	samples/rhythm/main.c

Name (Click to view in documentation)	Modules Used	Description	Location in Package
submit_album	Submit	Demonstrates editing an Album GDO and submitting it to Gracenote	samples/submit_album/main.c
submit_feature	Submit	Demonstrates processing music features and submitting a parcel to Gracenote	samples/submit_feature/main.c

## Reference Applications

Name	Modules Used	Description	Location in Package
MoodGrid_MoreLikeThis	Playlist	Demonstrates the MoreLikeThis use-case using Playlist and Mood.	reference_apps/MoodGrid_MoreLikeThis

### 2.2.4 Building a Sample Application

GNSDK provides sample applications for many common use cases. You can find the sample applications in the samples/<module> folders of your GNSDK package, where <module> corresponds to GNSDK modules and features, like musicid\_lookup\_album\_toc, musicid\_file\_albumid, and so on.

Each sample application folder contains the following files:

- main.c: C source for the sample application
- makefile: GNU Make makefile for building the sample application

Some modules may contain a /data subfolder if metadata exists for the sample application.

The GNSDK package also contains following makefiles:

- GNSDK Build files, sample\_vars.mk, gather\_sources.mk, and rules\_samples.mk
- Platform specific makefiles in the /builds directory

Each sample application makefile includes sample\_vars.mak and rules\_sample.mk.

You can build sample applications on any of the GNSDK supported platforms.

### 2.2.4.1 Build Environment Requirements

GNSDK provides multi-threaded, thread-safe technology for a wide variety of platforms and languages. For a detailed list of these, see the Release Notes included in the software package.

### 2.2.4.2 Build Steps

To build and run a sample application:

1. Open a shell and navigate to the sample application folder.
2. Ensure that the proper environment variables are set for your platform.
3. Type make. This creates a **sample.exe** file (and generally other output files) in the same folder. Object files from the compiler are stored in an output folder.
4. Type the platform-specific command to run the sample.exe and include the ClientID, ClientID Tag, and License file information as command-line arguments as described below.

### Supported make commands

Command	Description
make	Builds a debug version
make release	Builds a release (non-debug) version

Command	Description
make release CFLAGS=-DUSE_LOCAL	Enables local database lookups

### 2.2.4.3 Including Client and License Information

Running the sample application generally requires the client ID, client ID tag, and license file provided with the SDK. Gracenote SDKs and sample applications cannot successfully execute without these values. You must supply them as command-line arguments to the sample application, in the following format:

```
./sample.exe <clientid> <clientid_tag> <license_file>
```

For example,

```
./sample.exe 12345678 ABCDEF0123456789ABCDEF012345679 "C:/gn_
license.txt"
```

### 2.2.4.4 Directing Output and Log Files

Each sample application output varies, but in general, you should see the application performing online querying and displaying scrolling output. Redirect the output to a file to capture it for viewing. The application also creates a sample.log file, which logs any errors that may have occurred.

### 2.2.4.5 Platform-Specific Build Instructions

This section contains important build notes for specific platforms.

#### Windows

Building on a Windows platform requires the following:

- Cygwin: For more information, see the file README\_windows.txt, included in the package.



- Visual Studio 2005 or later.

To build the sample application on a Windows platform:

1. Edit the cygwin.bat file to configure the correct variables for the installed Visual Studio version. (The default location for this file is generally C:\cygwin, but this is dependent on your specific configuration.) See examples below.
2. Follow the general instructions: *"Build Steps" on page 70.*
3. Be sure to navigate to the sample folder using the proper Cygwin path prefix of /cygdrive/c <or correct drive>/<path to sample>. For more information on using Cygwin, see the [Cygwin's User's Guide](#).

This example shows the original code delivered by Cygwin in the cygwin.bat file:

```
@echo off
```

```
C:
```

```
chdir C:\cygwin\bin
```

```
bash --login -i
```

This example shows how the code must be edited to call the correct variables for the installed Visual Studio version. Note that this code specifically calls Visual Studio 2008 and configures the variables before initializing Cygwin:

```
@echo off
```

```
:: Change this call based on the version of Visual Studio being used
```

```
:: If using Visual Studio 2008, call "%VS90COMNTOOLS%\."
```

```
:: If using Visual Studio 2005, call "%VS80COMNTOOLS%\."
```

```
call "%VS90COMNTOOLS%\..\VC\vcvarsall.bat" x86
```

```
C:
```

```
chdir C:\cygwin\bin
```

```
bash --login
```

## Windows CE

Because Windows CE does not have a relative path concept, the samples provided with GNSDK will not run without modifications. The following modifications are necessary:

- You must use an absolute path for the log files.
- You must set a path name for the database after SQLite is initialized, for example:

```
gnsdk_storage_sqlite_option_set(GNSDK_STORAGE_SQLITE_OPTION_
STORAGE_FOLDER, "/storage card/samples");
```

- You must set a path name for the database after Playlist is initialized, for example:

```
gnsdk_playlist_storage_location_set( "/storage card/samples");
```

The recommended way to run the samples is to create a .bat file with a line similar to this:

```
/storage card/samples_c/playlist/sample.exe 12345678
ABCDEF0123456789ABCDEF012345679 "\Storage card\samples\license.txt"
```

To run the samples:

1. Launch a command prompt.
2. Change directories (cd) to the running directory.
3. Invoke the .bat file to run the sample.

## Linux ARM

To build samples when using the Linux ARM-32 binaries:

1. Open a shell and navigate to the sample application folder:..../gnsdk\_<version>\_<date>/\_sample/<module>.
2. Type make clean ARCH=arm.
3. Type make ARCH=arm to build the executable file.

4. Navigate to the.../gnsdk\_<version>\_<date>/\_sample/linux\_arm-32 folder to access the gnsdk\_<module>\_sample.exe and run the sample.

## Linux and Solaris

Follow the general instructions in *"Build Steps" on page 70*.

## MacOS

Follow the general instructions in *"Build Steps" on page 70*.

To build samples when using Mac OS X x86\_64 binaries, use the command:

```
make ARCH=x86_64
```

### 2.2.5 GNSDK C Sample Application Walkthrough

This topic steps through a GNSDK C sample application. This application shows how to do music identification using the MusicID module. We chose this sample for a walkthrough because it demonstrates several tasks that are common to (or can be extrapolated to) most GNSDK product use cases.

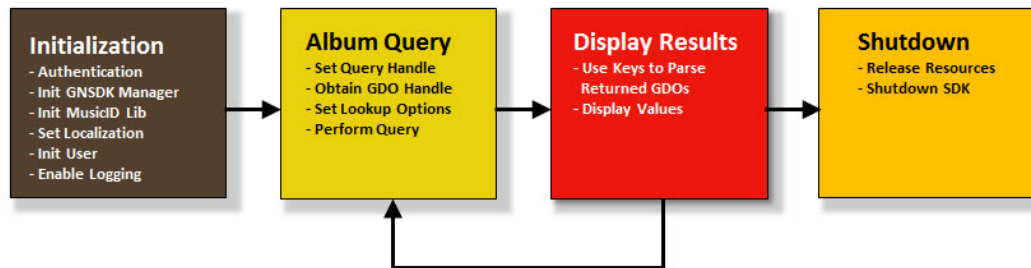
The sample application is called musicid\_gdo\_navigation/main.c and can be found in the /samples folder in the software package.

This application uses MusicID to look up Album content, navigate the hierarchy of data returned, and extract and display metadata results. As shown in the following image, most of the sample applications follow a similar pattern—authenticate/initialize, query Gracenote Service, parse and display results, and shut down.

This sample application:

- Initializes the GNSDK Manager, the MusicID library and local storage, enables logging, sets localization options, and registers the application user with the Gracenote Service.

- Performs album lookup queries using internal Gracenote identifiers.
- Parses returned Gracenote Data Objects (GDOs) and displays the results.
- Releases resources and shuts down the GNSDK when done.



In this topic, error handling is minimized and function calls are sometimes shown without some of their parameters. Refer to the sample application code to see how to implement error handling, and see the actual function call syntax.

#### 2.2.5.1 Albums That This Sample Queries On

The sample application queries the Gracenote Service for the following albums:

- Nelly - Nellyville
- Dido - Life for Rent
- Jean-Pierre Rampal - Portrait Of Rampal
- Various Artists - Grieg: Piano Concerto, Peer Gynt Suites #1
- Stephen Kovacevich - Brahms: Rhapsodies, Waltzes & Piano Pieces
- Nirvana - Nevermind
- Eminem - Encore
- Japanese album: <Japanese title>
- Chinese album: <Chinese title>

The following shows the metadata displayed for one album.

**Sample album output:**

```
*****Sample MusicID Query*****
Match.

***Navigating Result GDO***
Album:
  Package Language: English
  Credit:
    Contributor:
      Name Official:
        Display: Nirvana
      Origin Level 1: North America
      Origin Level 2: United States
      Origin Level 3: Washington
      Origin Level 4: Seattle
      Era Level 1: 1990's
      Era Level 2: 1990's
      Era Level 3: 1990's
      Artist Type Level 1: Male
      Artist Type Level 2: Male Group
  Title Official:
    Display: Nevermind
  Year: 1991
  Genre Level 1: Alternative & Punk
  Genre Level 2: Alternative
  Genre Level 3: Grunge
  Album Label: Geffen
  Total In Set: 1
  Disc In Set: 1
  Track Count: 12
  Track:
    Track TUI: 12845136
    Track Number: 1
    Title Official:
      Display: Smells Like Teen Spirit
    Year: 1991
  Track:
    Track TUI: 2897701
    Track Number: 2
    Title Official:
      Display: In Bloom
  Track:
    Track TUI: 12845138
    Track Number: 3
    Title Official:
      Display: Come As You Are
```

```
Year: 1991
Track:
  Track TUI: 2897703
  Track Number: 4
  Title Official:
    Display: Breed
Track:
  Track TUI: 2897704
  Track Number: 5
  Title Official:
    Display: Lithium
Track:
  Track TUI: 2897705
  Track Number: 6
  Title Official:
    Display: Polly
Track:
  Track TUI: 2897706
  Track Number: 7
  Title Official:
    Display: Territorial Pissings
Track:
  Track TUI: 2897707
  Track Number: 8
  Title Official:
    Display: Drain You
Track:
  Track TUI: 2897708
  Track Number: 9
  Title Official:
    Display: Lounge Act
Track:
  Track TUI: 2897709
  Track Number: 10
  Title Official:
    Display: Stay Away
Track:
  Track TUI: 2897710
  Track Number: 11
  Title Official:
    Display: On A Plain
Track:
  Track TUI: 2897711
  Track Number: 12
  Title Official:
```

Display: Something In The Way / [Hidden Track]

### 2.2.5.2 Prerequisites

To build and run any of the sample applications, see *"Building a Sample Application" on page 69*. In general, you need three things from Gracenote:

1. License File—Details your permissions and access to GNSDK libraries.
2. Client ID—Your specific GNSDK access identifier.
3. Client Tag—A hashed representation of the Client ID that guarantees its authenticity.

These three items are used to authenticate your application's access to MusicID as well as other GNSDK libraries. You pass them when you invoke a sample application program at the command-line:

```
> sample.exe <clientid> <clientidtag> <licensefile>
```

### 2.2.5.3 Initialization

After `main()` does some initial error checking on the Client ID, Client Tag and License File parameters, it calls `_init_gnsdk()` which makes a number of initialization calls.

#### Initialize the GNSDK Manager

Your application must initialize the GNSDK Manager prior to calling any other GNSDK library. The GNSDK Manager is the central controller for your application's interaction with the Gracenote Service. The initialization call returns a GNSDK Manager handle for use in subsequent SDK calls, including any other initialization calls. This is a required call.

```
/*  
 * Initialize the GNSDK Manager  
 */  
gnsdk_manager_handle_t sdkmgr_handle= GNSDK_NULL;  
gnsdk_manager_initialize(&sdkmgr_handle, license_path, GNSDK_MANAGER_  
LICENSEDATA_FILENAME);
```

The `GNSDK_MANAGER_LICENSEDATA_FILENAME` define is passed for the parameter that is supposed to indicate the license string length but, instead, indicates that the license data should be read from a file.

## Enable SDK Logging

The sample application enables SDK logging in the `_enable_logging()` function. The sample application does not write to this log file, only the SDK. The log file this generates can help Gracenote in debugging your application. The `_enable_logging()` function makes the following GNSDK call:

```
gnsdk_manager_logging_enable(  
/* Log file path */  
"sample.log",  
/* Include entries for all packages and subsystems */  
GNSDK_LOG_PKG_ALL,  
/* Include only error and warning entries */  
GNSDK_LOG_LEVEL_ERROR|GNSDK_LOG_LEVEL_WARNING,  
/* All logging options: timestamps, thread IDs, and so on. */  
GNSDK_LOG_OPTION_ALL,  
/* Max size of log: 0 means a new log file is created each run */  
  
0,  
/* GNSDK_TRUE = old logs will be renamed and saved */  
GNSDK_FALSE  
);
```

## Initialize Storage and Local Lookup

To open a database you must first create a configuration for it. The configuration will set the path and the access mode to the database. The following snippet will set a configuration for a database, and configure it to support MusicID text queries and images:

```
gnsdk_cstr_t db_identfier_id = "sampleDB_ID";  
gnsdk_cstr_t gracenote_db_path = ".././sampledb_path";  
gnsdk_cstr_t access_mode = GNSDK_LOOKUPDATABASE_ACCESS_READ_WRITE;  
gnsdk_config_handle_t config_handle = GNSDK_NULL;  
  
/* Error checking has been removed for brevity. Always check return  
codes from GNSDK functions. */
```



```
gnsdk_config_create(&config_handle);
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_ALL_
LOCATION, gracenote_db_path);
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_
ACCESS, access_mode);

/* configure the database to support MusicID-Text queries and MusicID
images */
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_
ENABLE_MUSICID_TEXT, GNSDK_VALUE_TRUE);
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_
ENABLE_MUSICID_IMAGES, GNSDK_VALUE_TRUE);
```

Once you have the configuration set, open the database, as shown in the following snippet:

```
gnsdk_lookupdatabase_open(db_identfier_id, config_handle);
```

### Initialize the MusicID Library Manager

Next, the application then initializes its interaction with the Gracenote MusicID library. Every GNSDK library must be initialized before an application can successfully call any of its APIs.

```
/*
 * Initialize the MusicID Library
 */
gnsdk_musicid_initialize(sdkmgr_handle);
```

### Initialize the User Handle

Every application user is required to register with the Gracenote Service. To perform queries, an application must first register a new user and get its handle. A

user represents an individual installation of a specific Client ID. This ensures that each application instance is receiving all required metadata entitlements. Users are represented in GNSDK by their handles. These handles contain the client ID/Tag string. The `_init_gdsdk()` function calls `_get_user_handle()` to either create a new user handle or restore a user handle from serialized storage.

After your application creates a new user, it should save its handle to serialized storage, where it can be retrieved every time your application needs it to use again. If an application registers a new user on every use instead of storing a serialized user, then the user quota maintained for the Client ID is quickly exhausted. Once the quota is reached, attempting to create new users will fail. To maintain an accurate usage profile of your application, and to ensure that the services you are entitled to are not being used unintentionally, it is important that your application registers a new user only when needed, and then stores that user for future use.

To register as a new user, your application can call `gnsdk_manager_user_register()`, then save the serialized user information in a file for future use.

For user registration and serialization, your application uses the following two calls:

- **`gnsdk_manager_user_register()`**—Registers the user and creates serialized data for storage and later retrieval.
- **`gnsdk_manager_user_create()`**—Creates the user handle from the registered user.

### ***Local-Only Registration***

Your application has the option to register for 'local-only' queries with the following call:

**`gnsdk_manager_user_register(GNSDK_USER_REGISTER_MODE_LOCALONLY)`**

In this case, the returned User object is not registered with the online service. A 'local-only' user can ONLY perform local queries and a failure will occur if an attempt is made to do an online query.

If the above call, is followed with a **gnsdk\_manager\_user\_set\_autoregister()** call, then, when the SDK does its first only query with a 'local-only' user, online registration takes place and the User object is automatically updated. The resulting User object can then be subsequently used for both online and local queries.

### Initialize Localization

Finally, this sample application makes the following localization calls in the `_set_locale()` function. Note that these calls can be done at anytime, but must be done *after* user registration, since they require a user handle parameter.

- **gnsdk\_manager\_locale\_load()**—Sets locale and creates a locale handle for subsequent calls. GNSDK locales are identifiers to a specific set of lists in the Gracenote Service. By using a locale, an application instructs the Gracenote Service to return only the data contained in a specific list. A locale is defined by a language and (optionally) a list region, a list descriptor, and a group. this sample application sets language to English (GNSDK\_LANG\_ENGLISH) and the region to default (GNSDK\_REGION\_DEFAULT).
- **gnsdk\_manager\_locale\_set\_group\_default()**—Sets a locale's global group default; all GDOs will use this locale unless specifically overwritten by `gnsdk_manager_gdo_set_locale()`. If the default is not set, no locale-specific results would be available. The locale group was set in the local handle with the previous call when the GNSDK\_LOCALE\_GROUP\_MUSIC was passed.

#### 2.2.5.4 Queries

An application can make a MusicID identification query in several ways, including text lookups, TOC lookups, fingerprint lookups, and so on. For a complete list of these options and examples, see *"MusicID Overview" on page 10*.

After identifying an element, Gracenote recommends using GDOs (Gracenote Data Objects) to do additional navigation and retrieve metadata. For information about GDOs, see *"About Gracenote Data Objects (GDOs)" on page 138*.

However, in addition to GDOs, there are several unique identifier types that can access Gracenote media elements. To learn more about these non-GDO Gracenote identifiers, see *"Working with Non-GDO Identifiers" on page 159*.



**Note:** This sample application (`musicid_gdo_navigation`) uses unique identifiers called TUIs to perform the initial identification (lookup) query. TUI stands for "Title Unique Identifier" and is used for internal Gracenote identification. Compared to text lookups, TOC lookups, and fingerprint lookups, TUIs are rarely used. They are used here as a convenient way to find a specific Album.

## Album Lookup

After initialization, this sample application calls `_do_sample_tui_lookups()` to perform a number of album lookups. In turn, `_do_sample_tui_lookups()` calls `_perform_sample_album_tui_lookup()`.

In `_perform_sample_album_tui_lookup()`, the sample application makes the following GNSDK calls to query a specific album:

- **`gnsdk_musicid_query_create()`**—Create the query handle.
- **`gnsdk_manager_gdo_create_from_id()`**—Obtain GDO handle. This takes the TUI and TUI tag parameters that uniquely identify an album. Under the hood, this creates a stub GDO that is used, in turn, for the query method.
- **`gnsdk_musicid_query_set_gdo()`**—Add the stub GDO from the last call to the query handle.
- **`gnsdk_musicid_query_option_set()`**—Add the options for local lookup to the query handle.
- **`gnsdk_musicid_query_find_albums()`**—Perform the query.

## GDO Navigation and Display

Each GNSDK identification query returns a GDO. GDO fields indicate how many matches were returned (none, single match, or multiple matches), which varies based on query criteria.

GDO metadata results can be either partial or full. Partial metadata may be sufficient for applications that only need to display basic information to the end user. Other applications may need full results. In all cases, applications should check to see if the GDO contains partial or full results and then determine if the partial is sufficient. For more information about full and partial results, see *"About Gracenote Data Objects (GDOs)" on page 138*. For a list of partial results returned in GDOs, see the GNSDK Data Dictionary.

### ***Parsing the Returned Response GDO***

A query returns a Response GDO containing a child GDO for each match. Since we are querying on a specific Gracenote Identifier (TUI and TUI tag), we are only going to get one child GDO containing full metadata results. However, the code parses the returned GDO to handle the multiple matches possibility. This is done in the `perform_sample_album_tui_lookup()` function:

```
/*
 * Find number of matches using the GNSDK_GDO_VALUE_RESPONSE_RESULT_
COUNT value key
 */
gnsdk_manager_gdo_value_get(response_gdo, GNSDK_GDO_VALUE_RESPONSE_
RESULT_COUNT, 1, &value_string);
album_count = value_string ? atoi(value_string) : 0;

/*
 * Get child if album_count > 0 using the GNSDK_GDO_CHILD_ALBUM child
key
 */
gnsdk_manager_gdo_child_get(response_gdo, GNSDK_GDO_CHILD_ALBUM, 1,
&album_gdo);

/*
 * Find if child GDO contains full or partial results using the GNSDK_
GDO_VALUE_FULL_RESULT value key.
 */
gnsdk_manager_gdo_value_get(album_gdo, GNSDK_GDO_VALUE_FULL_RESULT, 1,
&value_string);

/*
 * If this GDO only contains partial metadata (atoi(value_string) ==
0), then re-query for the full results.
 * First, however, set match back to the existing query handle then
```

```
release the selected match as the
 * query handle has it.
 */
gnsdk_musiconid_query_set_gdo(query_handle, album_gdo);
gnsdk_manager_gdo_release(response_gdo);

/*
 * Query for this match in full
 */
gnsdk_musiconid_query_find_albums(query_handle, &match_type, &response_gdo);
```

### ***Parsing the Child GDO***

Once we have a Response GDO with a child GDO containing full results for our album query, we can then extract the child GDO values and display them. This is done in the `_navigate_album_response_gdo()` function, also called in `_perform_sample_album_tui_lookup`.

The query to get full results also returns a Response GDO. All queries return Response GDOs. This means that the `_navigate_album_response_gdo()` function has to repeat the call to get the child album GDO (see the code sample below). This time, however, the application is aware that the child GDO contains full results and does not need to re-test for this.

```
/*
 * Get the child album GDO containing full results from the album
 * Response GDO
 */
gnsdk_manager_gdo_child_count(response_gdo, GNSDK_GDO_CHILD_ALBUM,
&child_count);
if (child_count > 0)
{
gnsdk_manager_gdo_child_get(response_gdo, GNSDK_GDO_CHILD_ALBUM, 1,
&album_gdo);
...
}
```

After getting the child GDO, `_navigate_album_response_gdo()` does the following:

1. Extracts and displays values (metadata) from the GDO using value keys.
2. Calls functions to navigate additional child GDOs (sub-objects). These functions extract and display metadata values from those objects.

### ***Extract Metadata and Display***

The navigate album function calls `_display_gdo_value()` to extract and display a metadata value. The display function calls `gnsdk_manager_gdo_value()` to extract values from the GDO using defined value keys, for example, `GNSDK_GDO_VALUE_YEAR`, `GNSDK_GDO_VALUE_GENRE_LEVEL_1`, `GNSDK_GDO_VALUE_ALBUM_TRACK_COUNT`, and so on.

Your application does not need to free GDO values and they will remain valid until the GDO handle is released.

### ***Navigate and Parse Additional Child GDOs***

In addition to `_navigate_album_response_gdo()`, which parses and displays album metadata, additional functions are also called to parse child GDOs (sub-objects):

- `_navigate_track_gdo()` - Parse and display track GDO metadata.
- `_navigate_credit_gdo()` - Parse and display credit GDO metadata.
- `_navigate_title_official_gdo()` - Parse and display official title GDO metadata.
- `_navigate_contributor_gdo()` - Parse and display contributor GDO metadata.
- `_navigate_name_official_gdo()` - Parse and display official name GDO metadata.

### **2.2.5.5 Releasing Resources and Shutting Down**

Before the program exits, it calls `_shutdown_gnsdk()`, which releases the user handle and shuts down the MusicID library, local storage and the GNSDK:

```
gnsdk_manager_user_release(user_handle);

/*
 * The sample app code (not shown) saves the serialized user handle
 * for later use during initialization.
 */

/* Shutdown the manager to shutdown all libraries */
gnsdk_manager_shutdown();
```

`gnsdk_manager_shutdown()` shuts down all "`_initialize`" functions.

It is a best practice to release resources when they are no longer needed, for example:

```
/* Release GDO (in _navigate_name_official_gdo()) */
gnsdk_manager_gdo_release(name_official gdo);

/* Release Query Handle (in _perform_sample_album_tui_lookup()) */
gnsdk_musicid_query_release(query_handle);
```





## Chapter 3 Develop and Implement

This section describes how to develop and implement GNSDK applications.

### 3.1 Application Flow

This topic describes how to get started with GNSDK development.

Every GNSDK application follows this general design and application flow:

1. Include the GNSDK header files for the modules your application requires. These should be based on the Gracenote products you licensed, such as MusicID.
2. Include other application headers that your application requires.
3. Initialize Manager.
4. Initialize other modules as needed, such as MusicID.
5. If you are using a Gracenote local database, initialize it.
6. Enable Logging. Gracenote recommends that the logging functionality of GNSDK be enabled to some extent. This aids in application debugging and issue reporting to Gracenote.
7. Create a User (or deserialize an existing User) to get a User handle. All queries require a User handle with correct Client ID information to perform the query.
8. Create a query handle with lookup criteria and any query options.
9. Perform the query. Gracenote returns a response Gracenote Data Object (GDO) with full or partial results. The GDO may contain a single match or multiple matches, requiring additional queries to refine the results. See *"About Gracenote Data Objects (GDOs)" on page 138*.
10. Process the query result to get to the desired element and retrieve its metadata.
11. Perform clean up by releasing all GDO and query handles.

12. Release the User handle.
13. Shutdown the GNSDK module and the GNSDK Manager.

## 3.2 Working with Header Files and Libraries

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 of the optional modules should be used.

Each GNSDK module has one or more corresponding header files. Other header files are also provided for public types and error codes. To include the necessary headers for a GNSDK application to run, simply include **gnsdk.h**, which automatically include headers for all modules.

If you wish to exclude headers for those modules that you will not be using, you may define certain values that the SDK uses to determine which modules to include. This is optional, and is only necessary if you do not wish to provide the GNSDK's pre-defined headers for unused modules. The following list contains the values that can be defined, and what they refer to. All are turned on by default:

- **GNSDK\_DSP**: Interface for the DSP APIs
- **GNSDK\_LINK**: Interface for Link content APIs
- **GNSDK\_LOOKUP\_LOCAL**: Interface for the Lookup Local APIs
- **GNSDK\_LOOKUP\_LOCALSTREAM**: Interface for the Lookup Local Stream APIs
- **GNSDK\_MANAGER**: Interface for the GNSDK Manager APIs
- **GNSDK\_MOODGRID**: Interface for Mood Grid APIs
- **GNSDK\_MUSICID**: Interface for MusicID APIs
- **GNSDK\_MUSICID\_FILE**: Interface for MusicID File APIs
- **GNSDK\_MUSICID\_MATCH**: Interface for MusicID Match APIs
- **GNSDK\_MUSICID\_STREAM**: Interface for MusicID Stream APIs
- **GNSDK\_PLAYLIST**: Interface for the Playlist APIs
- **GNSDK\_RHYTHM**: Interface for the Rhythm APIs

- **GNSDK\_STORAGE\_SQLITE**: Interface for the SQLite APIs
- **GNSDK\_SUBMIT**: Interface for the Submit APIs
- **GNSDK\_VIDEO**: Interface for VideoID APIs

These values must be defined in your application prior to including **gnsdk.h**, as shown in the example below:

```
#define GNSDK_VIDEO 1
#define GNSDK_STORAGE_SQLITE 1
#define GNSDK_LINK 1
#include "gnsdk.h" //Include all applicable GNSDK headers
```

If you do not define any of these values, the headers for all modules will be included with your source. Including all headers is the easiest way to ensure that all appropriate headers are included in your application.

### 3.3 Authorizing GNSDK Applications

Gracenote manages access to metadata using a combination of product licensing and server-side metadata entitlements.

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

Gracenote uses all of this information to create a unique customer ID (called a client ID), a license file, and server-side metadata entitlements that are tailored to your application goals.

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. The client ID is used by Gracenote Media Services to enable access to the metadata your application is entitled to use.

All client IDs 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 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.

### 3.3.1 License Files

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. License file information is provided to `gnsdk_manager_initialize()` when initializing the GNSDK. Below is an example license file, showing enabled and disabled Gracenote products for Customer A. These products generally map to corresponding GNSDK modules.

```
-- BEGIN LICENSE v1.0 ABC123XYZ --
licensee: Gracenote, Inc.
name: CustomerA
notes: [CustomerA Account#]
client_id: 9999999
musicid_text: enabled
musicid_cd: enabled
playlist: enabled
local_images: enabled
local_mood: enabled
-- SIGNATURE ABC123XYZ --
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890abcdefghijklmnopqrstuvwxyz
-- END LICENSE ABC123XYZ --
```

### 3.3.2 Client IDs

Each GNSDK customer receives a unique client ID/Tag 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/Tag string consists of two sets of numbers separated with a hyphen. The number before the hyphen is considered the 'ID' and the number after, the 'Tag'. A client ID/Tag string has the following format:

```
<10 character client ID>-<17 character client ID tag>
```

When registering a new User with `gnsdk_manager_user_register()`, you must pass in the client ID and client ID tag as separate parameters. To create a user handle, use the serialized data created during registering, plus the client ID tag with the function `gnsdk_manager_user_create()`.

It is best practice to save the serialized user data, created during registration, to a file, and then to use the content of that file for later creations of user handles.

For more information see the section Users.

### 3.3.3 Users

To perform queries, an application must first register a new User and get its handle. A User represents an individual installation of a specific client ID/Tag string. This ensures that each application instance is receiving all required metadata entitlements.

Users are represented in GNSDK by User handles. You can create a User handle using `gnsdk_manager_user_register()` and `gnsdk_manager_user_create()`.

When creating a new User with `gnsdk_manager_user_create()`, you must pass in the serialized user string created with `gnsdk_manager_user_register()` and the client ID tag. For more information about client IDs and client ID tags, see *"Client IDs" on the previous page*.

Each application installation should only request a new User once and store the serialized representation of the User for future use. If an application registers a new User on every use instead of storing a serialized User, then the User quota maintained for the client ID is quickly exhausted. Once the quota is reached, attempting to create new Users will fail. If an application uses a single User registration across multiple installations of the application—in short, forcing all the installations to use the same User—the application risks exhausting Gracenote per-user query quota.

To maintain an accurate usage profile of your application, and to ensure that the services you are entitled to are not being used unintentionally, your application should register new Users only when needed, and then store that User for future queries.

To store users once they are created, you specify a callback function that the SDK calls when a User handle needs to be stored locally. To specify the callback, use `gnsdk_manager_user_set_autoregister()` and pass to it the associated `client_ID`.

The SDK triggers the callback function when a change to the User handle occurs. Applications should store serialized User data at a location that can persist between application launches (for example, local file storage). In the callback, you save the user data to persistent storage. This simplifies handling of user data, and ensures data storage occurs at appropriate times.

The callback also allows an application to register a new user in offline mode and then register it for online use during the first online query. In this instance, the callback is triggered after the online query is performed.

Basic User management process:

1. First application run: register new User and get serialized string.
2. Create the user handle from serialized user string and client ID.
3. Specify a callback function to handle User storage and registration.
4. Provide User handle to GNSDK APIs that require one.
5. Release User handle when finished.
6. Subsequent application runs: create User handle from stored serialized data.

For example:

```
gnsdk_user_handle_t user_hdl = GNSDK_NULL; // User handle
gnsdk_str_t          ser_str  = GNSDK_NULL; // Serialized user string

/* Create serialized user string */
gnsdk_manager_user_register(GNSDK_USER_REGISTER_MODE_ONLINE, client_
id, client_id_tag, client_app_version, &ser_str);

/* Call API to register and create new user. This API takes a
serialized user
```

```
    string ('ser_str') (either created new or retrieved from
    storage and creates a non-serialized user handle in 'user_hdl'
*/
gnsdk_manager_user_create(ser_str, client_id, &user_hdl);

/* Register callback function to store serialized user data */

gnsdk_manager_user_set_autoregister(user_hdl, _user_store_callback,
    &callback_data);

/* Perform queries */
/* ... */

/* Release user */

error = gnsdk_manager_user_release(user_hdl);

/* Close GNSDK */

/* callback function */

static void _user_store_callback(gnsdk_void_t* callback_data,
    gnsdk_cstr_t serialized_user)
{
/* Code to store serialized user string for later reuse. */

save_serialized_user_data(serialized_user); /* NOT a Gracenote API */
}
```

## 3.4 Initializing and Shutting Down GNSDK

### 3.4.1 *Initializing an Application*

Your application needs to initialize a module before using it. It needs to first call `gnsdk_manager_initialize()` to use the GNSDK Manager module. This call



requires a client ID and a Gracenote license file and returns a handle necessary to initialize other modules.

Depending on your application's logic, you may need to retain the GNSDK Manager handle to initialize other modules from different locations in your application. One option is to globally manage the GNSDK manager handle so it is always available. Alternatively, you can call `gnsdk_manager_initialize()` multiple times when needed.

#### 3.4.1.1 Specifying the License File

Your application must provide the license file on the first (and likely only) call to `gnsdk_manager_initialize()`. This API's `license_data` and `license_data_len` parameters give you the following options when doing this:

- **Memory buffer**—Set the `license_data` parameter to the memory buffer pointer, and the `license_data_len` to the memory buffer size.
- **Null-terminated string**—Set the `license_data` parameter to the string buffer pointer, and the `license_data_len` to `GNSDK_MANAGER_LICENSEDATA_NULLTERMSTRING`.
- **Filename**—Set the `license_data` parameter to a string buffer containing the relative filename of a file containing the license data, and the `license_data_len` to `GNSDK_MANAGER_LICENSEDATA_FILENAME`.
- **Stdin**—Set the `license_data` parameter to `GNSDK_NULL`, and the `license_data_len` to `GNSDK_MANAGER_LICENSEDATA_STDIN`.



**Note:** You have the option to call `gnsdk_manager_initialize()` again to overwrite an existing license file with a new license file.

#### 3.4.2 Shutting Down an Application

Calls to the initialize API on any GNSDK module are counted. The first call to initialize the module does the actual work, and every subsequent call to initialize only increments an initialization count. So, calling initialize multiple times is safe and not resource-intensive.

For each module you initialize, you can either call the shutdown for that module or just call GNSDK Manager shutdown. The latter call will shut down all libraries. Shutting down a specific module may save you some memory during program execution, but, other than that, there is no advantage in shutting down a specific module. If resources associated with a module have not been released, then its shutdown call will not work. The module will continue to be alive until the GNSDK Manager shutdown call is made.



**Note:** Do not call a shutdown function if the corresponding initialize function returns an error.

### **3.4.3** *Example: Initializing and Shutting Down*

Sample Application: `musicid_lookup_album_text/main.c`

This application shows the initialization and shutdown of a GNSDK application, including a simple album lookup based on text.

## **3.5** Setting Network and Proxy Options

An application can set options to aid in working with networks and proxy servers. Depending on the topology of your network, you can alter certain values using `gnsdk_manager_user_option_set()` so your Gracenote application can effectively work within your system.

### **3.5.1** *Proxy Server Options*

GNSDK supports the ability to set proxy server values such as host name, port numbers, and user IDs. The following table contains options available to `gnsdk_manager_user_option_set()` that support use of a proxy server:

Option	Use
GNSDK_USER_OPTION_PROXY_HOST	Value for this option is a fully qualified host name with optional port number. The default port number is 80.
GNSDK_USER_OPTION_PROXY_USER	Value for this option is a valid user name for the proxy server. You do not need to set this option if a user name is not required.
GNSDK_USER_OPTION_PROXY_PASS	Value for this option is the valid password for the proxy server. You do not need to set this option if a password is not required.

The following calls set options for the proxy server:

```
gnsdk_manager_user_option_set(
    user_handle,
    GNSDK_USER_OPTION_PROXY_HOST,
    "http://proxy.example.com:8080/"
);
gnsdk_manager_user_option_set(
    user_handle,
    GNSDK_USER_OPTION_PROXY_USER,
    "proxyUserName"
);
gnsdk_manager_user_option_set(
    user_handle,
    GNSDK_USER_OPTION_PROXY_PASS,
    "proxyPW"
);

/*
 * To set an option as global to all users, pass in NULL
 * for the user handle. Options set globally apply only
 * to user handles created after the global option is set.
 */

gnsdk_manager_user_option_set(
    GNSDK_NULL,
    GNSDK_USER_OPTION_PROXY_HOST,
    proxy_host);
```

## 3.5.2

### 3.5.3 *Network Options*

GNSDK supports the ability to set options for network timeouts, load balancing, or interface types. The following table contains network options available to `gnsdk_manager_user_option_set()`:

Option	Use
GNSDK_USER_OPTION_NETWORK_TIMEOUT	Sets the network timeout for all GNSDK queries. Value for this option is a string with a numeric value representing the number of milliseconds to set for network timeouts. The default internal timeout is 30 seconds. If the network connection is weak or lost during communication, the SDK does not attempt a retry. The user application must make all retry attempts.
GNSDK_USER_OPTION_NETWORK_LOADBALANCE	Enables distributing queries across multiple Gracenote co-location facilities. When not enabled, queries will resolve to a single co-location. You must ensure that any security settings, such as a firewall, in your network infrastructure do not affect outbound access and prevent GNSDK from transmitting queries to various hosts with unique IP addresses,
GNSDK_USER_OPTION_NETWORK_INTERFACE	This option provides NIC support, such as WIFI and GSM. For devices with multiple Network Interfaces, you can set which NIC is used by Gracenote online features for all queries and per query, respectively. The NIC is selected through its IP address.

### 3.5.4 *Testing Connection Ability*

You can test your applications ability to connect to the Gracenote Service by calling `gnsdk_manager_test_gracenote_connection()`. This is a quick way to see if your application has been properly set up.

## 3.6 Configuring Logging

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

1. **Enable GNSDK logging** - This creates log file(s) that you and Gracenote 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 \*\_logging\_write() APIs to add your application's log entries to the GNSDK logs.
3. **Implement your own logging mechanism (via the logging callback).**

The most typical use case for GNSDK logging is to configure a single log file to capture all logged messages and errors from both your application and the SDK.

### 3.6.1 Enabling GNSDK Logging

To enable Gracenote SDK logging, use the gnsdk\_manager\_logging\_enable() function. For example:

```
/*
 * Enable SDK logging
 */
gnsdk_manager_logging_enable(
    "sample.log",          // File path
    GNSDK_LOG_PKG_GNSDK,  // Include all GNSDK package IDs, including
                           // your app's ID
    GNSDK_LOG_LEVEL_ALL,  // Include all level entries (warning, info,
                           // debug and error)
    GNSDK_LOG_OPTION_ALL, // All logging options
    0,                    // Max size of log in bytes: 0 means a new log file
                           // will be created each run
    GNSDK_FALSE           // GNSDK_TRUE = old logs will be renamed and
                           // saved
);
```

The GNSDK\_LOG\_OPTION\_ALL define encompasses all logging options which include category (info, debug, warning and error), timestamp, source information

(file and line number), package ID, thread ID, and if logging should be done asynchronously. See the API reference for more information.

The GNSDK\_LOG\_PKG\_GNSDK define means that all GNSDK package IDs will be logged. Use the GNSDK\_LOG\_PKG\_GCSL if you just want all low-level Gracenote Client Standard Library (GCSL) package IDs. Use GNSDK\_LOG\_PKG\_ALL for both GNSDK and GCSL package IDs. In most cases, GNSDK\_LOG\_PKG\_GNSDK should be sufficient.

The GNSDK\_LOG\_LEVEL\_ALL options means that all levels (error, warning, info, and debug) will be logged. There are options available if you just want messages for specific levels.

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.

### **3.6.2** *Adding to GNSDK Logs*

Your application can write to the GNSDK logs using the gnsdk\_manager\_logging\_write() and gnsdk\_manager\_logging\_vwrite() functions. Use the latter function to pass a variable number of parameters - see the API reference for more information.

The SDK allows an application to register one or more of its own package IDs into the GNSDK logging system with the gnsdk\_manager\_logging\_register\_package() call. The application can then enable, disable or filter its own logging messages based on the registered package IDs. For example:

```
/*
 * Log entries are identified by the 'package' that logs them. Here we
 * generate
 * a package ID for our application to allow us to log entries of our
 * own in the GNSDK log
 * The package ID must be in the range GNSDKPKG_ID_APP_START to MAX_
 * GNSDK_PKG_ID
 */
#define MYAPP_PACKAGE_ID    GNSDKPKG_ID_APP_START+0x01

/*
 * Register our package ID with the logging system - this will make
```

```
our logging
* entries show up with the package description of "Sample App"
*/
gnsdk_manager_logging_register_package(MYAPP_PACKAGE_ID, "Sample
App");

/*
* Sample write to log file
*/
gnsdk_manager_logging_write(
    150,          // Source line number(optional)
    "main.c",     // Source file (optional)
    MYAPP_PACKAGE_ID, // Package ID of application making call
    GNSDK_LOG_LEVEL_ERROR, // Error mask for this logging message
    "Error info: %s", // Error message format
    info          // Error message format argument
);
```

The logging system determines the applicability of a log message for each enabled log, and, if appropriate, writes a log message to multiple enabled logs.

### **3.6.3** *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 Unix Syslog or the Windows Event Log.

Enabling callback is done with the `gnsdk_manager_logging_enable_callback()` API. The logging callback can be enabled alongside, or instead of, file logging. Your application can call this API multiple times with different Package IDs to enable multiple Package IDs. The package and filter will apply only to the logging message sent to the callback. To disable the logging callback, call `gnsdk_manager_logging_enable_callback` with the callback parameter set to `GNSDK_NULL` for each enabled PackageID.

**For example:**

```
/*
* Register our callback with the GNSDK Manager
```

```
*/
gnsdk_manager_logging_enable_callback(
    _logging_callback_fn,
    GNSDK_NULL,          // Optional data passed to the callback
    GNSDK_LOG_PKG_ALL,   // Messages from all libraries
    GNSDK_LOG_LEVEL_ALL, // Message of all levels
    GNSDK_LOG_OPTION_ALL // All logging options: timestamps, thread
IDs, etc
);

...

static gnsdk_void_t GNSDK_CALLBACK_API
_logging_callback_fn(
    gnsdk_void_t*   user_data,
    gnsdk_uint16_t  package_id,
    gnsdk_uint32_t  mask,
    gnsdk_cstr_t    format,
    va_list         argptr
)
{
    /*
     * Map the GNSDK level to the Syslog level and pass along the
message
    */
    if (mask & GNSDK_LOG_LEVEL_ERROR)
    {
        vsyslog(LOG_ERR, format, argptr);
    }
    else if (mask & GNSDK_LOG_LEVEL_WARNING)
    {
        vsyslog(LOG_WARNING, format, argptr);
    }
    else if (mask & GNSDK_LOG_LEVEL_INFO)
    {
        vsyslog(LOG_INFO, format, argptr);
    }
    else if (mask & GNSDK_LOG_LEVEL_DEBUG)
    {
        vsyslog(LOG_DEBUG, format, argptr);
    }
} /* _logging_callback_fn() */
```



## 3.7 Implementing Status Callbacks

GNSDK allows you to implement status callback functions that are called at certain points during ongoing operations. Implementing a callback function allows you to monitor an operation's status, and cancel it if necessary. You can set a callback for any of the following operations:

- MusicID queries
- Enriched content fetches
- Locale and List loading and updates
- Mood presentation handle creation
- Submit parcel handle creation
- Video queries

For example, when creating a MusicID query, you can register a status callback, which GNSDK calls at different stages. This allows you to display query progress to users or respond to various conditions.

### 3.7.1 *Displaying Operation Status*

To create a status callback, implement a function with the following prototype:

```
typedef gnsdk_void_t  
(GNSDK_CALLBACK_API *gnsdk_status_callback_fn) (  
    gnsdk_void_t*    user_data,  
    gnsdk_status_t    status,  
    gnsdk_uint32_t    percent_complete,  
    gnsdk_size_t      bytes_total_sent,  
    gnsdk_size_t      bytes_total_received,  
    gnsdk_bool_t*     p_abort  
);
```

When this function is called, the status parameter contains a value indicating operation status, such as `gnsdk_status_begin`, `gnsdk_status_progress`, or `gnsdk_status_complete`. For example, a status callback might respond to the following statuses to display operation progress:

```
switch(status)
{
    case gnsdk_status_begin:
        printf("\nBegin...");
        break;
    case gnsdk_status_connecting:
        printf("\nConnecting...");
        break;
    case gnsdk_status_sending:
        printf("\nSending...");
        break;
    case gnsdk_status_receiving:
        printf("\nReceiving...");
        break;
    case gnsdk_status_progress:
        printf("\nIn progress...");
        break;
    case gnsdk_status_complete:
        printf("\nComplete\n");
        break;
    default:
        break;
}
```

For a full list of status values, see “Status Callbacks” in the GNSDK C API Reference.

### **3.7.2** *Registering a Status Callback*

A status callback is registered when you create a query or load or update a locale or list. The following functions accept a status callback as a parameter:

- gnsdk\_musid\_query\_create()
- gnsdk\_link\_query\_create()
- gnsdk\_manager\_locale\_load()
- gnsdk\_manager\_locale\_update()
- gnsdk\_manager\_list\_retrieve()
- gnsdk\_manager\_list\_update()
- gnsdk\_moodgrid\_presentation\_create()

- `gnsdk_submit_parcel_create()`
- `gnsdk_video_query_create()`

### 3.7.3 *canceling Operations*

To cancel a query or large operation that is in progress, you can set the status callback's `p_abort` parameter to `GNSDK_TRUE`. In response, the query returns an abort error.

### 3.7.4 *Example: Using a Callback*

Sample Application: `musicid_file_libraryid/main.c`

This example demonstrates using a status callback function for a MusicID query.

## 3.8 Using Locales

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. It is less complicated (but less flexible) than 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. For information about lists, see *"Using Lists" on page 117*.

### 3.8.1 *Loading a Locale*

To load a locale, use the `gnsdk_manager_locale_load()` function. As can be seen in the sample below, Locale properties are:

- **Group** Group type of locale such as Music or EPG 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

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).

To configure the locale:

- Set the group key to the respective GNSDK\_LOCALE\_GROUP\_\*.
- Set the language key (GNSDK\_LANG\_\*) to the required language.
- Set the region and descriptor keys to the respective GNSDK\_\*\_DEFAULT key.
- Set the user handle. User handles are required. However, for locales (and lists) the responses are not tied to the individual user handle. All users have access to locally available locales and lists.

For example:

```
gnsdk_manager_locale_load(  
    GNSDK_LOCALE_GROUP_MUSIC,      // Group - Music (others include EPG,  
    and Video)  
    GNSDK_LANG_ENGLISH,           // Language - English  
    GNSDK_REGION_DEFAULT,         // Default is US (others include China,  
    Japan, Europe, and so on)  
    GNSDK_DESCRIPTOR_DETAILED,    // Default music descriptor is  
    'detailed' (versus 'simplified')  
    user_handle,                  // User handle
```

```

GNSDK_NULL,          // No status callback
GNSDK_NULL,          // No status userdata
&locale_handle       // Locale handle to be set
);

```

### 3.8.1.1 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
Europe (EU)	Europe	Europe
China (CN)	China and Taiwan	China

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

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.

### **3.8.2** *Locale Groups*

Setting the locale for a group causes the given locale to apply to a particular media group, such as Music 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.

The locale group property can be set to one of the following values:

- `GNSDK_LOCALE_GROUP_MUSIC`: Sets the locale for all music-related objects
- `GNSDK_LOCALE_GROUP_PLAYLIST`: Sets the locale for playlist generation
- `GNSDK_LOCALE_GROUP_VIDEO`: Sets the locale for all video-related objects
- `GNSDK_LOCALE_GROUP_EPG`: Sets the locale for all EPG-related objects

Once a locale has been loaded, you must call one of the following functions to set the locale before retrieving locale-dependent values from a GDO:

- `gnsdk_manager_locale_set_group_default()`: This function sets a default locale. When a locale is set to be the default, it becomes the default locale for its inherent group. So, you can set a default for each locale group, such as Music, EPG, and so on. The default locale is automatically applied to each new GDO (that is relevant for that locale group). Setting a locale manually for a GDO (using `gnsdk_manager_gdo_set_locale`) overrides the default locale.
- `gnsdk_manager_gdo_set_locale()`: This function sets the locale of the locale-dependent data for a specific GDO handle. Note that this function

does not set the default locale for a group. To set the default locale, you must use the `gnsdk_manager_locale_set_group_default()` function.

### **3.8.3** *Locale-Dependent Values and List Types*

The table below summarizes locale-dependent value keys and their corresponding list types. The list type values actually returned depend on the type of GDO you are working with. You can load lists using `gnsdk_manager_gdo_set_locale()`.

List types are categorizations of related list metadata. For example, `GNSDK_LIST_TYPE_MOODS` contains a hierarchical list of moods for audio metadata, such as Blue (Level 1) and Earthy/Gritty/Soulful Level 2).

#### **3.8.3.1** *Locale-Dependent Genre Levels*

The Gracenote Genre System provides a locale-dependent view of the genre hierarchy based on the user's geographic location or cultural preference. This allows you to deliver localized solutions for consumers in different parts of the world. Localized solutions allow representation and navigation of music in a manner that is expected in that region.

For example, consumers in the U.S. would expect to find Japanese or French Pop music in a World genre category, while North American Pop would be expected to be labeled as Pop. In Japan, consumers would expect to find Japanese Pop under Pop and French and North American Pop under Western Pop. In a solution shipped globally, all Pop music would be categorized as Pop, regardless of the origin of the music.

### 3.8.3.2 Music Locale-Dependent Values and List Types

Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_ARTISTTYPE_LEVEL1	GNSDK_LIST_TYPE_ARTISTTYPES
GNSDK_GDO_VALUE_ARTISTTYPE_LEVEL2	
GNSDK_GDO_VALUE_COMPOSITION_FORM	GNSDK_LIST_TYPE_COMPOSITION_FORM
GNSDK_GDO_VALUE_ENTITY_TYPE	GNSDK_LIST_TYPE_CONTRIBUTORENTITYTYPES
GNSDK_GDO_VALUE_ERA_LEVEL1	GNSDK_LIST_TYPE_ERAS
GNSDK_GDO_VALUE_ERA_LEVEL2	
GNSDK_GDO_VALUE_ERA_LEVEL3	
GNSDK_GDO_VALUE_GENRE_LEVEL1	GNSDK_LIST_TYPE_GENRES
GNSDK_GDO_VALUE_GENRE_LEVEL2	
GNSDK_GDO_VALUE_GENRE_LEVEL3	
GNSDK_GDO_VALUE_INSTRUMENTATION*	GNSDK_LIST_TYPE_INSTRUMENTATION
GNSDK_GDO_VALUE_MOOD_LEVEL1	GNSDK_LIST_TYPE_MOODS
GNSDK_GDO_VALUE_MOOD_LEVEL2	
GNSDK_GDO_VALUE_ORIGIN_LEVEL1	GNSDK_LIST_TYPE_ORIGINS
GNSDK_GDO_VALUE_ORIGIN_LEVEL2	
GNSDK_GDO_VALUE_ORIGIN_LEVEL3	
GNSDK_GDO_VALUE_ORIGIN_LEVEL4	
GNSDK_GDO_VALUE_PACKAGE_LANGUAGE_DISPLAY	GNSDK_LIST_TYPE_LANGUAGES



Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_ROLE	GNSDK_LIST_TYPE_CONTRIBUTORS
GNSDK_GDO_VALUE_ROLE_CATEGORY	
GNSDK_GDO_VALUE_ROLE	GNSDK_LIST_TYPE_ROLES
GNSDK_GDO_VALUE_ROLE_CATEGORY	
GNSDK_GDO_VALUE_TEMPO_LEVEL1	GNSDK_LIST_TYPE_TEMPOS
GNSDK_GDO_VALUE_TEMPO_LEVEL2	
GNSDK_GDO_VALUE_TEMPO_LEVEL3	

\*GNSDK\_GDO\_VALUE\_INSTRUMENTATION will be removed in future releases.

### 3.8.3.3 Video Locale-Dependent Values

Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_AUDIENCE	GNSDK_LIST_TYPE_VIDEOAUDIENCE
GNSDK_GDO_VALUE_COLOR_TYPE	GNSDK_LIST_TYPE_VIDEOCOLORTYPES
GNSDK_GDO_VALUE_GENRE_LEVEL1	GNSDK_LIST_TYPE_GENRES_VIDEO
GNSDK_GDO_VALUE_GENRE_LEVEL2	
GNSDK_GDO_VALUE_GENRE_LEVEL3	
GNSDK_GDO_VALUE_KIND_TYPE	GNSDK_LIST_TYPE_VIDEOKINDTYPES
GNSDK_GDO_VALUE_REPUTATION	GNSDK_LIST_TYPE_VIDOREPUTATION
GNSDK_GDO_VALUE_SCENARIO	GNSDK_LIST_TYPE_VIDEOSCENARIO

Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_SERIAL_TYPE	GNSDK_LIST_TYPE_VIDEOSERIALTYPES
GNSDK_GDO_VALUE_SETTING_ENVIRONMENT	GNSDK_LIST_TYPE_VIDEOSETTINGENV
GNSDK_GDO_VALUE_SETTING_TIME_PERIOD	GNSDK_LIST_TYPE_VIDEOSETTINGPERIOD
GNSDK_GDO_VALUE_SOUND_TYPE	GNSDK_LIST_TYPE_VIDEOSOUNDTYPES
GNSDK_GDO_VALUE_SOURCE	GNSDK_LIST_TYPE_VIDEOSOURCE
GNSDK_GDO_VALUE_STORY_TYPE	GNSDK_LIST_TYPE_VIDEOSTORYTYPE
GNSDK_GDO_VALUE_STYLE	GNSDK_LIST_TYPE_VIDEOSTYLE
GNSDK_GDO_VALUE_TOPIC	GNSDK_LIST_TYPE_VIDEOTOPIC
GNSDK_GDO_VALUE_VIDEO_FEATURE_TYPE	GNSDK_LIST_TYPE_FEATURETYPES
GNSDK_GDO_VALUE_VIDEO_MOOD	GNSDK_LIST_TYPE_VIDEOMOOD
GNSDK_GDO_VALUE_VIDEO_MOOD	
GNSDK_GDO_VALUE_VIDEO_MOOD	
GNSDK_GDO_VALUE_VIDEO_PRODUCTION_TYPE	GNSDK_LIST_TYPE_VIDEOTYPES
GNSDK_GDO_VALUE_VIDEO_REGION	GNSDK_LIST_TYPE_VIDOREGIONS
GNSDK_GDO_VALUE_VIDEO_REGION_DESC	GNSDK_LIST_TYPE_VIDOREGIONS
GNSDK_GDO_VALUE_MEDIA_SPACE	GNSDK_LIST_TYPE_MEDIASPACES
GNSDK_GDO_VALUE_MEDIA_TYPE	GNSDK_LIST_TYPE_MEDIATYPES
GNSDK_GDO_VALUE_WORK_TYPE	GNSDK_LIST_TYPE_WORKTYPES

Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_RATING	GNSDK_LIST_TYPE_RATINGS
GNSDK_GDO_VALUE_RATING_DESC	
GNSDK_GDO_VALUE_RATING_TYPE	GNSDK_LIST_TYPE_RATINGTYPES
GNSDK_GDO_VALUE_RATING_TYPE_ID	GNSDK_LIST_TYPE_RATINGS

### 3.8.3.4 EPG Locale-Dependent Values

Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_EPGAUDIOTYPE	GNSDK_LIST_TYPE_EPGAUDIOTYPES
GNSDK_GDO_VALUE_EPGCAPTIONTYPE	GNSDK_LIST_TYPE_EPGCAPTIONTYPES
GNSDK_GDO_VALUE_EPGCATEGORY_L1	GNSDK_LIST_TYPE_IPGCATEGORIES_L1
GNSDK_GDO_VALUE_EPGCATEGORY_L2	GNSDK_LIST_TYPE_IPGCATEGORIES_L2
GNSDK_GDO_VALUE_EPGPRODUCTIONTYPE	GNSDK_LIST_TYPE_EPGPRODUCTIONTYPES
GNSDK_GDO_VALUE_EPGVIDEOTYPE	GNSDK_LIST_TYPE_EPGVIDEOTYPES
GNSDK_GDO_VALUE_EPGVIEWINGTYPE	GNSDK_LIST_TYPE_EPGVIEWINGTYPES

### 3.8.3.5 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.

### **3.8.4** *Updating Locales and Lists*

GNSDK supports storing locales and their associated lists locally. Storing locales locally improves access times and performance. Your application must include a database module (such as SQLite) to implement local storage. For more information, see *"Using SQLite for Storage and Caching" on page 133*.

Periodically, your application should update any locale lists that are stored locally. Currently, Gracenote lists are updated no more than twice a year. However, Gracenote recommends that applications update with `gnsdk_manager_locale_update()` or check for updates with `gnsdk_manager_locale_update_check()` every 14 days.

If new list revisions are available, `gnsdk_manager_locale_update()` immediately downloads them, but `gnsdk_manager_locale_update_check()` does not. This makes `gnsdk_manager_locale_update_check()` suitable for applications that wish to limit network traffic during normal operation and defer downloading new revisions until a greater capacity network connection is available.

As an alternative to explicitly calling the check functions, you can instead register a callback function to be called when a list or locale has gone out of date. Use the function `gnsdk_manager_list_update_notify()` to register a callback for when a list goes out of date, and `gnsdk_manager_locale_update_notify()` for a locale. Your application can use the callback to either update the list or locale, or just take note that it needs to be done.

If the SDK infers your locale lists are out of date, it returns a `GNSDKERR_ListUpdateNeeded` error code. This error is only returned if your application attempts to access metadata via a response GDO that cannot be resolved.





**Note:** Updates require the user option `GNSDK_USER_OPTION_LOOKUP_MODE` to be set to `GNSDK_LOOKUP_MODE_ONLINE` (default) or `GNSDK_LOOKUP_MODE_ONLINE_ONLY`. This allows the



SDK to retrieve lists from the Gracenote service. You may need to toggle this option value for the update process. For more information about setting the user option, see *"Setting Local and Online Lookup Modes" on page 122*.

### 3.8.5 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.
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.  <div>  <b>Note:</b> 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. </div>

Practice	Description
Set a <i>persistence</i> flag when updating. If interrupted, repeat update.	<p>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.</p> <p>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.</p>
Call <code>gnsdk_manager_storage_compact()</code> after updating lists or locales.	<p>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 <code>gnsdk_manager_storage_compact()</code> to remove these.</p> <div>  <p><b>Note:</b> 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 <code>gnsdk_manager_storage_compact()</code> during startup or shutdown after an update has finished.</p> </div>

### 3.8.6 Example: Loading a Locale and Retrieving Locale-Sensitive Metadata

Sample Application: `musicid_lookup_matches_text/main.c`

This application finds matches based on input text and loads a locale which is used by GNSDK to provide appropriate locale-sensitive metadata for certain metadata values.

### 3.8.7 Using Lists

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

Lists are either *flat* or *hierarchical*. Flat lists contain only one level of metadata, such as languages. Hierarchical lists are tree-like structures with parents and children.

Typically, hierarchical lists contain general display strings at the upper levels (Level 1) and more granular strings at the lower levels (Level 2 and Level 3, respectively). For example, a parent Level 1 music genre of Rock contains a grandchild Level 3 genre of Rock Opera. The application can determine what level of list granularity is needed by using the list functions (gnsdk\_sdkmgr\_list\_\*) in the GNSDK Manager. For more information, see "*Core and Enriched Metadata*" on page 5.

Lists can be specific to a *Region*, as well as a *Language*. For example, the music genre known as J-pop (Japanese pop) in America is called pop in Japan.

In general, Lists provide:

- Mappings from Gracenote IDs to Gracenote Descriptors, in various languages
- Delivery of content that powers features such as MoreLikeThis, Playlist, and Mood



**Note:** MoreLikeThis, Playlist, and Mood also use *Correlates*. These lists specify the correlations among different genres, moods, and so on. For example, Punk correlates higher to Rock than it does to Country. Using the MoreLikeThis feature when playing a Punk track will likely return more Rock suggestions than Country.

### 3.8.7.1 List and Locale Interdependence

GNSDK provides *Locales* as a way to group lists specific to a region and language. Using locales is relatively straightforward for most applications to implement. However, it is not as flexible as directly accessing lists - most of the processing is done in the background and is not configurable. For more information, see "*Using Locales*" on page 106.

Conversely, the List functionality is robust, flexible, and useful in complex applications. However, it requires more implementation design and maintenance. It also does not support non-list-based metadata and the GDO value keys (GNSDK\_GDO\_VALUE\_\*).

For most applications, using locales is more than sufficient. In cases where the application requires non-locale functionality (for example, list display), implementing both methods may be necessary. The following sections discuss each method's advantages and disadvantages.

The GNSDK locale functionality includes:

- Loading a locale into the application using `gnsdk_manager_gdo_load_locale()`. GNSDK loads a locale behind the scenes, and loads only those lists that are not already in memory. When database (such as SQLite) cache is enabled, the GNSDK Manager automatically caches the list. When caching is not enabled, GNSDK downloads the locale for each request.
- You can set a loaded locale as the application's default locale, and this can eliminate the need to call `gnsdk_manager_set_locale()` for a GDO . Instead, you call the default locale.
- Setting a locale for a GDO using `gnsdk_manager_gdo_set_locale()`. Doing this ensures that all language-specific metadata is returned with the language (and region and descriptor, if applicable) defined for the locale.

When using locales, be aware that:

- You can serialize locales and lists and save them within your application for later re-use. To store non-serialized locales and lists, you must implement a database cache. If you do not want to use a cache, you can instead download required locales and lists during your application's initialization.
- A list's contents cannot be displayed (for example, to present a list of available genres to a user).
- Performing advanced list functionality, such as displaying list items in a dropdown box for a user's selection, or accessing list elements for filtering lookups (for example, restricting lookups to a particular mood or tempo), is not possible.



The GNSDK list functionality includes:

- Directly accessing individual lists using the `gnsdk_manager_list_*` APIs.
- Accessing locale-specific list metadata.

When using lists, be aware that:

- There are numerous list handles.
- List handles must be released after use.
- Locale-specific non-list metadata is not supported.
- List metadata GDO keys are not supported.
- When using `gnsdk_manager_list_retrieve()` to get a list, you must provide a user handle.

### 3.8.7.2 Retrieving Locale Information

To retrieve the valid locale configurations available in your local lists cache, use the `gnsdk_manager_locale_available_get()` function. Locale configurations are combinations of values that you can use to set the locale for your application. The function returns the following values:

- Group
- Language
- Region
- Descriptor

To get each locale configuration, use the `gnsdk_manager_locale_available_count()` function to provide ordinals to the `gnsdk_manager_locale_available_get()` function.

For example, the following code retrieves the available locale configurations:

```
gnsdk_locale_handle_t locale_handle = GNSDK_NULL;
gnsdk_cstr_t type = GNSDK_NULL;
gnsdk_cstr_t region = GNSDK_NULL;
gnsdk_cstr_t descriptor = GNSDK_NULL;
gnsdk_cstr_t language = GNSDK_NULL;
```

```
gnsdk_uint32_t count = 0;
gnsdk_uint32_t ordinal = 0;
gnsdk_error_t error = GNSDK_SUCCESS;

error = gnsdk_manager_locale_available_count(&count);

if (!error)
{
    for(ordinal=1; ordinal<=count; ordinal++)
    {
        error = gnsdk_manager_locale_available_get(
            ordinal, &type, &language,
            &region, &descriptor);
    }
}
```

You can use the returned values directly in the `gnsdk_manager_locale_load()` function to load a locale. For more information about loading locales, see *"Using Locales" on page 106*.

### 3.8.7.3 Updating Lists

To update lists, follow the procedure described in *"Updating Locales and Lists" on page 115*. The following list-specific functions are available:

- `gnsdk_manager_list_update()`
- `gnsdk_manager_list_update_check()`
- `gnsdk_manager_list_update_notify()`
- `gnsdk_manager_locale_update_notify()`

### 3.8.7.4 Example: Accessing a Music Genre List

This example demonstrates accessing a list and displaying the list's elements.

Sample application: `musicid_image_fetch/main.c`

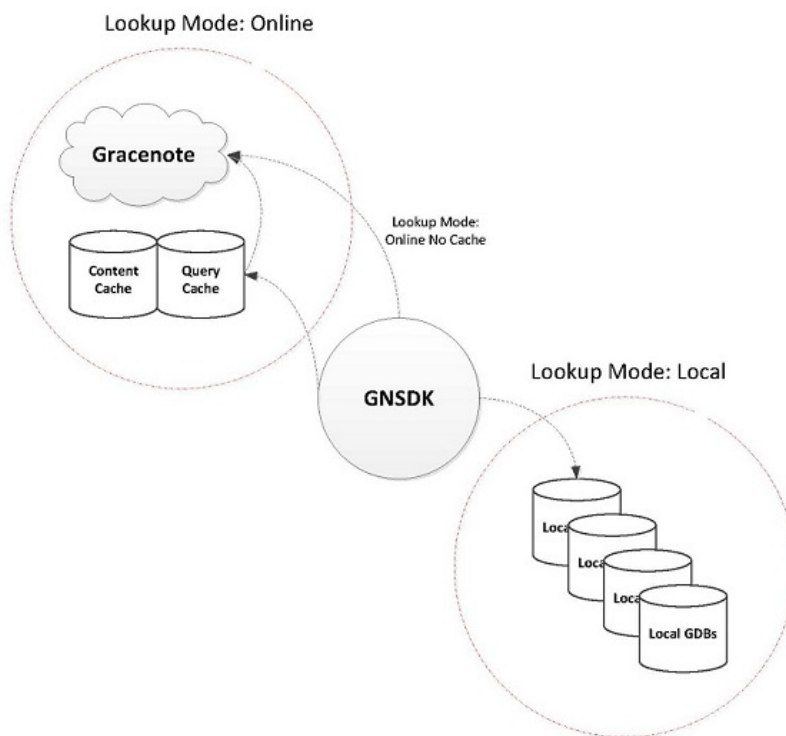
This application demonstrates how to access a genre list. It does a text search and finds images based on gdo type (album or contributor), and also finds images based on genre.

## 3.9 Local Storage

This section describes options for local storage and caching using GNSDK.

### 3.9.1 *Setting Local and Online Lookup Modes*

An application can perform local or online lookups as shown in the following diagram.



#### 3.9.1.1 Lookup Providers

A *lookup provider* is a module that implements the ability to query data for matches. GNSDK allows both local and online lookup providers to be enabled at run-time, and you can use lookup modes to configure how they will be used.

If the application supports an online connection, by default, GNSDK enables an online lookup provider that connects to the Gracenote Service. To enable other lookup providers, you can initialize a `gnsdk_lookup` module. For example, to enable the local lookup provider, call the `gnsdk_lookup_local_initialize()` function.

### 3.9.1.2 Storage Providers

A *storage provider* is a module that implements storage for the entire SDK. GNSDK allows various storage providers to be enabled at run-time.

By default, no storage provider is enabled. To enable a storage provider, you can initialize a `gnsdk_storage` module. For example, to enable the SQLite storage provider, call the `gnsdk_storage_sqlite_initialize()` function.



**Note:** The only supported storage provider included with GNSDK for Auto is SQLite.

### 3.9.1.3 Lookup Modes

GNSDK allows you to determine whether lookups will be done locally or online, by setting *lookup modes*. The lookup mode options allow an application to switch between the two main lookup providers (local and online). GNSDK is designed to operate exactly the same way in either mode, providing identical behavior whether operating locally or online.

GNSDK supports the following lookup mode options:

- `GNSDK_LOOKUP_MODE_ONLINE`: This is the default lookup mode except as noted below. Queries are performed against Gracenote Service online. If the application has called `gnsdk_storage_sqlite_initialize()`, the local cache is checked first. If the result is not found in the cache, the query goes online and writes the result to the local cache.
- `GNSDK_LOOKUP_MODE_ONLINE_NOCACHEREAD`: Queries are performed against Gracenote Service online. If the application has called `gnsdk_storage_sqlite_initialize()`, online query results are written to the local cache.

- **GNSDK\_LOOKUP\_MODE\_ONLINE\_CACHEONLY:** Queries are performed against the local cache. This mode requires that the application first calls `gnsdk_storage_sqlite_initialize()`.
- **GNSDK\_LOOKUP\_MODE\_ONLINE\_NOCACHE:** Queries are performed against Gracenote Service online.
- **GNSDK\_LOOKUP\_MODE\_LOCAL:** This is the default lookup mode if your Gracenote license file supports local queries only. Queries are performed against enabled local databases. This requires `gnsdk_lookup_local_initialize()` and `gnsdk_storage_sqlite_initialize()` to have been called.



**Note:** While supported, using a cache is not recommended and is not best practice. For online lookups, use `GNSDK_LOOKUP_MODE_ONLINE_NOCACHE`. This is the standard online lookup mode. While `GNSDK_LOOKUP_MODE_ONLINE_CACHEONLY` and `GNSDK_LOOKUP_MODE_ONLINE_NOCACHEREAD` are available, using them can damage performance.

For example, the following call sets the user lookup option so that lookups are performed locally:

```
gnsdk_manager_user_option_set(  
    user_handle,  
    GNSDK_USER_OPTION_LOOKUP_MODE,  
    GNSDK_LOOKUP_MODE_LOCAL  
);
```

Your application can use `GNSDK_USER_OPTION_CACHE_EXPIRATION` option to set the length of time before a cache lookup times out.

#### 3.9.1.4 Local and Online Lookup Comparison

The following tables indicate whether technologies, content, and metadata are supported for local and online lookups.



**Note:** Note: Local lookups support album and contributor matching. Online lookups support album matching only. No contributor matches will occur online.

Recognition/Discovery Technology	Local/Online Support
MusicID-CD	Local and online
MusicID Stream	Online only
MusicID text-based recognition	Local and online
MusicID-File (text)	Local and online
MusicID-File (fingerprint)	Online only
Playlist	Local only

Link Content	Local/Online Support
Cover art	Local and online
Artist images	Local and online
Genre art	Local only

Metadata	Local/Online Support
Genre	Local and online
Origin	Local and online
Era	Local and online
Artist Type	Local and online
Mood	Local and online
Tempo	Online only

Metadata	Local/Online Support
Classical	Local and online

### 3.9.2 Working with Local Databases

Working with local databases is very similar to working with online Gracenote services. In both cases, queries are set up in the same way, and results are processed in the same way.

#### 3.9.2.1 Initializing a Local Database

To work with a local database, you must first initialize the SQLite library and then initialize the local lookup library:

```
gnsdk_storage_sqlite_initialize(sdkmgr_handle);  
gnsdk_lookup_local_initialize(sdkmgr_handle);
```

Once initialized, open databases using configuration handles.

To open a database you must first create a configuration for it. The configuration will set the path and the access mode to the database. The following snippet will set a configuration for a database, and configure it to support MusicID text queries and images:

```
gnsdk_cstr_t db_identfier_id = "sampleDB_ID";  
gnsdk_cstr_t gracenote_db_path = "../..../sampledb_path";  
gnsdk_cstr_t access_mode = GNSDK_LOOKUPDATABASE_ACCESS_READ_WRITE;  
gnsdk_config_handle_t config_handle = GNSDK_NULL;  
  
/* Error checking has been removed for brevity. Always check return  
codes from GNSDK functions. */  
  
gnsdk_config_create(&config_handle);  
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_ALL_  
LOCATION, gracenote_db_path);  
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_  
ACCESS, access_mode);  
  
/* configure the database to support MusicID-Text queries and MusicID  
images */
```

```
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_
ENABLE_MUSICID_TEXT, GNSDK_VALUE_TRUE);
gnsdk_config_value_set(config_handle, GNSDK_CONFIG_LOOKUPDATABASE_
ENABLE_MUSICID_IMAGES, GNSDK_VALUE_TRUE);
```

Once you have the configuration set, open the database, as shown in the following snippet:

```
gnsdk_lookupdatabase_open(db_identfier_id, config_handle);
```

The system accesses this database, along with any other local database, for all local queries. GNSDK will close the database on shutdown.

You can set your cache 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.

### 3.9.2.2 Setting Local Lookup Mode

To do local lookups for all queries, you can set the user lookup option or set the query lookup option to indicate that the local database should be used for all lookups. For more information about the user lookup option, see *"Setting Local and Online Lookup Modes" on page 122*.

```
gnsdk_manager_user_option_set(
    user_handle,
    GNSDK_USER_OPTION_LOOKUP_MODE,
    GNSDK_LOOKUP_MODE_LOCAL
);
```

You can also set or override on a query by query basis in other libraries. For example, the following code will set local lookup in the Music ID File library:

```
/* To use the local data stores for a query, set the 'use local
lookup'
* query option to true.
* By default, the lookup will be handled online but many applications
* will want to start with a local query and then fall back to an
online
* query for anything not matched. See the "musicid_lookup_album_local_
online"
* sample for an example of this 'hybrid' approach.
```



```
*/  
gnsdk_musicidfile_query_option_set(  
    query_handle,  
    GNSDK_MUSICIDFILE_OPTION_LOOKUP_MODE,  
    GNSDK_LOOKUP_MODE_LOCAL  
);
```



**Note:** For MusicID Radio, `gnsdk_radioid_channel_option_set()` behaves similarly. However, `find_tracks` is only supported online and logos are supported local only. You cannot switch a handle that is doing track queries in the background to local mode. It is considered best practice that every time the radio station is changed, create a new channel handle and at that time, the logo should be fetched and kept in memory. Then set the channel to online, call `audio_start`, and handle track events.

### 3.9.2.3 Getting Manifest Information about Local Databases

GNSDK provides local databases, which differ based on region, configuration, and other factors. 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.

To get database information, use `gnsdk_lookupdatabase_info_get()` to retrieve data into an information GDO, from which you can access data via keys. Consult your Gracenote representative for further information.

In the following example, `gnsdk_lookupdatabase_info_get()` is used to retrieve stored album cover art sizes.

GNSDK provides album cover art, and artist and genre images in different sizes. You can retrieve the available image sizes by using the `GNSDK_GDO_VALUE_IMAGE_SIZE` key on the storage GDO. This allows you to request images in available sizes only, rather than spending cycles requesting image sizes that are not available.

The following code retrieves the available image sizes, and stores them in a local variable `image_size`.

```
gnsdk_uint32_t    count        = 0;
gnsdk_gdo_handle_t info_gdo    = GNSDK_NULL;
gnsdk_gdo_handle_t store_gdo   = GNSDK_NULL;
gnsdk_cstr_t      storage_type = GNSDK_NULL;
gnsdk_cstr_t      image_size   = GNSDK_NULL;

gnsdk_lookupdatabase_info_get("example", &info_gdo);
gnsdk_manager_gdo_child_count(info_gdo, GNSDK_GDO_CHILD_STORAGE_INFO,
&count);

for (gnsdk_uint32_t ord = 1; ord <= count; ord++)
{
    gnsdk_manager_gdo_child_get(info_gdo, GNSDK_GDO_CHILD_STORAGE_INFO,
ord, &store_gdo);
    gnsdk_manager_gdo_value_get(store_gdo, GNSDK_GDO_VALUE_STORAGE_TYPE,
1, &storage_type);
    if (gcs1_string_equal(storage_type, GNSDK_LOOKUP_LOCAL_STORAGE_
CONTENT, GNSDK_TRUE))
    {
        gnsdk_manager_gdo_value_get(store_gdo, GNSDK_GDO_VALUE_IMAGE_SIZE,
1, &image_size);
    }
}
```

Once you have retrieved the available image sizes, you can set the image size for a query using the `gnsdk_link_query_option_set()` function. For more information about retrieving images, see *"Accessing Enriched Content using Asset Fetch" on page 223*.

#### 3.9.2.4 Using Results Databases

GNSDK applications supports application created read/write databases for storing local and online query and image fetch results. These databases are referred to as **Results databases**. Results databases are similar to the off-the-shelf monthly production gdb's in that they have the same files and schema and are configured, enabled and opened in the same way. The main differences are that they are created by the application and are read-write. An application can have

multiple Results databases open at one time, depending on the need of the application. These databases store results that can be returned in future queries.

#### Results Databases:

- Support incremental Gracernote database updates that can eliminate the need to do a base replace of the entire database with the latest information.
- Support deferred (or tethered) online searches when a user attempts a search without a connection.
- Control disk space used by Gracernote databases.
- Do not support record deletion or expiration.
- Do not require the existence of a Gracernote delivered Production database in an application.

### Application Flow

The process of using a Results database is similar to using any other GNSDK application. Each database must have a user supplied ID. This ID supports the use of a specific database.

1. Typically initialize the GNSDK, provide license information and create a user.
2. Open databases as necessary. Each database requires a unique ID and config\_handle.
3. Query and update as required.
4. Manage database sizes, as necessary. This might be needed only when data is added to a database.
5. Close databases and application.

### ***Managing Disk Space***

The Results database might need to be controlled to help maintain appropriate disk usage and content. If you are opening an existing Results database and intend to add to it, checking disk usage is a good step to take immediately after

adding information. In addition, you can't delete individual pieces of data from within a Results database, so if it is necessary to retire and purge old data, you must delete and recreate the entire database. The following snippet will check the size of a database, and if it is too large, it deletes it.

```
/* Retrieve database size based on database ID */
gnsdk_lookupdatabase_size_get( db_identfier_id, &size );

/* Check if DB size is larger than a pre-determined value.  If so,
   Remove old DB and start new one.  */
if (size > MAX_DB_SIZE)
{
    /* Close the database and delete it. */
    gnsdk_lookupdatabase_close( db_identfier_id );
    gnsdk_lookupdatabase_delete( config_handle );

    /* code to reopen database here */
    ...
}
```

### 3.9.3 Querying and Updating a Results Database

Whenever a query occurs, such as with `gnsdk_musicid_query_find_matches()`, all open databases are searched. There are no specific functions for a specific Results database search.

Gracenote Data Objects (GDOs) can be added to a Results database using the function `gnsdk_lookupdatabase_record_add()`, and images can be added with `gnsdk_lookupdatabase_image_add()`.



**Note:** If the same result is found in more than one open database, the result with the higher revision will be returned. Also, while you can always add a query result to a Results database, if it is already present with a revision that is equal to or higher than that of the one you are adding, the add will not take place.

## Results Database Functions

The GNSDK provides several functions that allow you to open, read, write, or query databases. When making a database query, all open databases will be searched. You do not specify which database you wish to use. However, the API provides the following functions that allow you to use a specific database, based on the ID that you provided for it:

Function	Use
<code>gnsdk_lookupdatabase_open()</code>	Open local database set - database set is set of related Gracernote databases. This call will create new or open existing database set based on configuration given.
<code>gnsdk_lookupdatabase_close()</code>	Close lookup local database - will wait if any threads have outstanding queries with this database. If this API is not called, all open databases will be closed on <code>gnsdk_manager_shutdown()</code> .
<code>gnsdk_lookupdatabase_record_add()</code>	Adds a record to the database.
<code>gnsdk_lookupdatabase_image_add()</code>	Adds an image to the database.
<code>gnsdk_lookupdatabase_size_get()</code>	Retrieves database size.
<code>gnsdk_lookupdatabase_delete()</code>	Deletes the local database files. Only lookup related GDB files are deleted.
<code>gnsdk_lookupdatabase_validate()</code>	Validates a database.
<code>gnsdk_lookupdatabase_info_get()</code>	Retrieves database information that can be used for debugging purposes.

.

## Examples: Results Databases

Sample Application: `musicid_automotive/main.c`

This application shows the initialization and shutdown of a GNSDK application, the opening of a Production database, and one Results database. It also shows database information lookup of the Results database, queries, and updates.

Sample Application: `musicid_persist_and_manage/main.c`

This more complex application shows the initialization and shutdown of a GNSDK application, the opening of a Production database, and two Results databases. In addition to queries and updates, it also shows size management of Results databases.

### 3.9.3.1 Using SQLite for Storage and Caching

The GNSDK SQLite (`gnsdk_storage_sqlite`) module provides a local storage solution using the SQLite database engine. This module is used to manage a local cache of content and Gracenote Service queries. This is for GNSDK use only - your application cannot use this database for its own storage. SQLite is the only database module available for Gracenote Automotive.

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

Besides functions specific to SQLite, there are a set of general storage functions that apply to the database module. These general functions cover setting cache location and various cache maintenance operations (cleanup, validate, compact, flush, and so on). See the API Reference for a complete list.

Specifically, API calls are provided to manage 3 *stores* or *caches* (as indicated by the following defines):

1. `GNSDK_MANAGER_STORAGE_LISTS`—Stores Gracenote locale lists.
2. `GNSDK_MANAGER_STORAGE_CONTENTCACHE`—Stores cover art and related information.
3. `GNSDK_MANAGER_STORAGE_QUERYCACHE`—Stores media identification requests.



**Note:** Caching, with the exception of locales, is not a recommended practice. See [Setting Local and Online Lookup Modes](#) for more information.

To begin, your application needs to make the following call to initialize SQLite (after initializing GNSDK Manager and getting an SDK handle).

```
gnsdk_storage_sqlite_initialize(sdkmgr_handle);
```



**Note: Important:** It is possible to initialize this library at any time before or after other libraries have been operating. However, to ensure that all queries are properly cached, it should be initialized immediately after the GNSDK Manager and before any other libraries.

### Setting Location, Access, and File Size

You can set various storage attributes by creating a configuration handle. The following section of code shows setting the access mode to read mode, read/write mode, and location for gn\_lists.gdb:

```
gnsdk_config_handle_t h_config = GNSDK_NULL;
gnsdk_config_create(&h_config);

// access mode read only
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_LISTS_
ACCESS_MODE, GNSDK_CONFIG_MANAGER_STORAGE_ACCESS_READ_ONLY);

// access mode read/write
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_LISTS_
ACCESS_MODE, GNSDK_CONFIG_MANAGER_STORAGE_ACCESS_READ_WRITE);

// location
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_LISTS_
LOCATION, "./path/to/list/storage");

gnsdk_manager_storage_configure(h_config);
gnsdk_config_release(h_config);
```

You can set a maximum file size for temporary storage. If you do not want temporary storage, set the access mode to read only. In the following code, temporary storage values are set:

```
gnsdk_config_handle_t h_config = GNSDK_NULL;
gnsdk_config_create(&h_config);

// access mode - read only
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_TEMP_
ACCESS_MODE, GNSDK_CONFIG_MANAGER_STORAGE_ACCESS_READ_ONLY);

// access mode - read/write
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_TEMP_
ACCESS_MODE, GNSDK_CONFIG_MANAGER_STORAGE_ACCESS_READ_WRITE);

// location
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_TEMP_
LOCATION, "./path/to/temp/storage");

// file size
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_TEMP_
FILESIZE, "1024");
gnsdk_manager_storage_configure(h_config);
gnsdk_config_release(h_config);
```

Most content and query caches are controlled through GNSDK lookup mode. Configuration is limited to file size and location, as shown below.

```
gnsdk_config_handle_t h_config = GNSDK_NULL;
gnsdk_config_create(&h_config);

// file size
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_
QUERYCACHE_FILESIZE, "1024");
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_
CONTENTCACHE_FILESIZE, "1024");

// location
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_
QUERYCACHE_LOCATION, "./path/to/create/query_cache");
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_
CONTENTCACHE_LOCATION, "./path/to/create/content_cache");

gnsdk_manager_storage_configure(h_config);
gnsdk_config_release(h_config);
```



If your application needs all storage to reside in the same location, you can use `GNSDK_CONFIG_MANAGER_STORAGE_ALL_LOCATION` to set the location for all storages.

```
gnsdk_config_handle_t h_config = GNSDK_NULL;
gnsdk_config_create(&h_config);

// location
gnsdk_config_value_set(h_config, GNSDK_CONFIG_MANAGER_STORAGE_ALL_LOCATION,
    "./path/for/storages");

gnsdk_manager_storage_configure(h_config);
gnsdk_config_release(h_config);
```

As with all GNSDK "initialize" calls, there should be a corresponding "shutdown" call before your application exits:

```
gnsdk_storage_sqlite_shutdown();
```

### Linking to External SQLite Libraries

If you are using your own version of SQLite in your application in addition to the GNSDK instance of SQLite, you can link to it by using `gnsdk_storage_sqlite_use_external_library()`. This function indicates to SQLite that it should use a supplied SQLite3 shared library.

```
gnsdk_storage_sqlite_use_external_library(sqlite_filepath);
```

The parameter `sqlite_filepath` contains the filepath to the SQLite shared library.

This function must be called immediately before a call to `gnsdk_storage_sqlite_initialize()`.

After successful initialization, you next set a valid folder for local storage. This must be set before this library will successfully begin operating.

To ensure that all queries are properly cached, initialize the library immediately after the GNSDK Manager and before any other libraries.



**Note: Important:** Gracernote's instance of SQLite has been compiled with specific flags to enable optional features of SQLite. Your version of SQLite must be built with the following flags to be usable:

- `SQLITE_ENABLE_COLUMN_METADATA`
- `SQLITE_ENABLE_UNLOCK_NOTIFY`
- `SQLITE_OMIT_DEPRECATED`

### **3.9.4 Load Balancing**

GNSDK's load balancing capability enables it to automatically send its outgoing queries to a set of addresses. The purpose of this is to spread its queries across a set of host servers (hosts) so that no single host processes the entire load of a single SDK instance.

When load balancing is enabled, GNSDK retrieves a set of IP addresses for each hostname it connects to. These IPs include multiple Gracernote Service locations, rather than just a single IP (and likewise, location) as occurs when load balancing is disabled. Once accessed, GNSDK uses each IP in round-robin order when connecting to the specific hostname. This spreads the connections from a single SDK instance to multiple hosts.

GNSDK stores the set of IPs for each unique hostname for up to five (5) minutes before it re-retrieves the set. If a host stops responding during this time, GNSDK removes the IP from the set of IPs currently in use. After five minutes have elapsed, all IPs are refreshed. If all IPs currently in use prove bad (meaning, unable to be connected to), GNSDK immediately expires them and re-retrieves another set.

#### **3.9.4.1 Implementation Considerations**

Gracernote strongly recommends enabling this option for these application scenarios:

- The application performs long-running processes.

- The application performs large numbers (millions) of queries.

You may opt to not enabling the load balancing capability to minimize the occurrence of temporary metadata differences that may be present among various hosts, and access more consistent metadata results. However, be aware that metadata differences among hosts generally resolve within minutes.

### 3.10 About Gracenote Data Objects (GDOs)

The primary goal of any GNSDK application is to recognize media elements and access their metadata. When an application performs a query, Gracenote returns metadata about the target query element, such as the title and genre of an album element. In addition, information about the query operation is returned, such as its timestamp, the start and end range of the query results, and the number of additional results available from Gracenote.

GNSDK stores the information returned by a query within containers known as Gracenote Data Objects (GDOs). The contents of a GDO depends on:

- The kind of query and the search criteria, such as a CD TOC lookup, a fingerprint lookup, text lookup, and so on
- The query target element
- Information about the target element available from Gracenote

A GDO can contain values and/or other, related GDOs. GDOs have two purposes: Containers to access metadata returned from Gracenote, and as input to queries to retrieve additional metadata and other GDOs.

GDOs facilitate a key feature of GNSDK - interoperability between all of the Gracenote products and services. Results from one Gracenote query can be used as an input for another. For example, a MusicID result can immediately be used as an input for the Link module without the need for any application intervention. This interoperability is possible for nearly all combinations of Gracenote Services.

### 3.10.1 GDO Types

Every GDO has a type. For example, when an application performs a query to identify an Album, Gracenote returns a GDO of type Album. Therefore, for most applications, you can *infer* a GDO's type based on the target element of the query and knowing the underlying data model for the element.

If needed, your application can get the type of a GDO. For example, your application might request a GDO's type to confirm it matches the intended type.

Another use case is analyzing the results of a general text lookup. This kind of query can return multiple GDOs of *different* types. The application needs to process the results to determine which GDO is the best response to the query.

### 3.10.2 Response GDOs and Child GDOs

It is important to note that **every identification query returns a *response GDO***. A response GDO is a higher-level GDO typically containing these fields (from album response GDO):

- Match references (child GDOs)
- Needs decision flag (see *Matches That Require Decisions*)
- Matches range start
- Matches range end
- Total number of matches

As noted, a response GDO contains references to 0-n matches encapsulated in *child GDOs*. A child GDO is just like any other GDO once it is retrieved. It is not dependent on its parent GDO and has the same behaviors and features of other GDOs. The fact that it was once a child of another GDO does not matter.

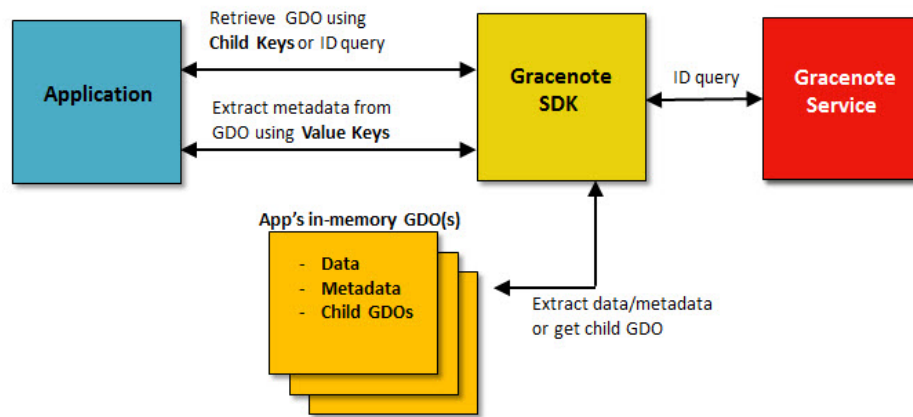
For example, an Album response GDO can contain child GDOs of type Track, or Artist, other Albums, and so on. A child GDO can contain its own child GDOs, such as Tracks, Artists, or Contributors, and so on.

A GDO's child objects are enumerated with an ordinal index, starting from 1 (not 0) for the first child object. Queries for child objects take this index as input.

### 3.10.3 Child Keys and Value Keys

To extract metadata from a GDO, or get a child GDO, your application must use defined keys. There are two kinds of keys: Value and Child.

- **Value Key**—Used to extract a specific piece of metadata from a GDO, for example GNSDK\_GDO\_VALUE\_ALBUM\_LABEL.
- **Child Key**—Used to get a child GDO, for example, GNSDK\_GDO\_CHILD\_TRACK.



### 3.10.4 Full and Partial Metadata Results

A response GDO from a query can return 0-n matches (child GDOs). These child GDOs 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 (for example, album titles) to the end user to make a selection. Once a selection is made, you can then do a secondary query to get the full results (if desired).

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 C API Reference*.

#### 3.10.4.1 Online and Local Database Queries

Applications that have an online connection can query Gracenote for information. Applications without an online connection, such as embedded applications used for stereo head units in cars, can instead query a Gracenote local database.

In general, local database queries return full GDO results, even when the response contains multiple matches.

Some online queries return partial GDO results containing just enough information to identify the matches. Using this information, the application can perform additional queries to obtain more information. In general, your application should *always* test GDO results to determine they are full or partial.

#### 3.10.5 Matches That Require Decisions

When your application makes an identification query, Gracenote returns a response GDO with one of the following:

- No GDO match
- Single GDO match
- Multiple GDO matches

In all cases, Gracenote returns high-confidence results based on the identification criteria. However, even high-confidence results could require additional decisions from a user.

In such cases, GNSDK sets `NEEDS_DECISION` to true, indicating that the end user should select the the final result. The application should present users with the returned match(es) and allow them to select (or reject) the match.

In general, GDO responses that need a decision by a user are:

- **Any multiple match response**
- **Any single match response** that Gracenote determines needs a decision by a user. Even though a single match might be good match candidate, Gracenote might determine that it is not quite perfect, based on the quality of the match and/or the criteria used to identify the match.

#### 3.10.5.1 About the Text Match Score

GNSDK provides a score (0-100) for text matches (accessible using the GNSDK\_GDO\_VALUE\_TEXT\_MATCH\_SCORE value key for an Album GDO) based on comparing the input text to the corresponding field in the match text. This score is *not* an indicator of text match *quality*.

For example, the result text could be substantially different from the input text because the input text contained a lot of incorrect information. Such a response indicates that the results have an amount of ambiguity that the application must resolve.

#### 3.10.5.2 About Video Product TOC Matches

VideoID uses several methods to perform TOC matches. This combination of matching methods enables client applications to accurately recognize media in a variety of situations.

Match Type	Description	Decision Needed?
Exact Match	Only one Product matches the DVD or Blu-ray TOC.	No
Multiple-Exact Match	Multiple Product matches exist for the DVD or Blu-ray TOC.	Yes
Title Match	An exact TOC match is not found, but one or more Product titles match.	Yes
Fuzzy Match	Match for media that has slight, known, and acceptable variations from well-recognized media.	Yes

### 3.10.6 GDO Workflows

A Gracenote identification query can return no matches, a single match, or multiple matches. The workflow for managing single and multiple GDO matches is similar, but not identical. The following sections describe these two workflows.

#### 3.10.6.1 GDO Workflow for a Single GDO Match

The simplest example of a GDO workflow is when an identification query returns a single match. For example, suppose there is a query to look up a track by name and find its containing album. If that Track only exists on one album, GNSDK identifies the single album and returns a response GDO that contains the core metadata for the identified album. The application can then access the metadata using value keys.

As described in [Matches that Require Decisions](#), some single GDO responses may require a decision by the application or end user. To address this possibility, the GDO workflow should include a test query using the GNSDK\_GDO\_VALUE\_RESPONSE\_NEEDS\_DECISION value key to determine if Gracenote has pre-determined that the GDO response needs a decision.

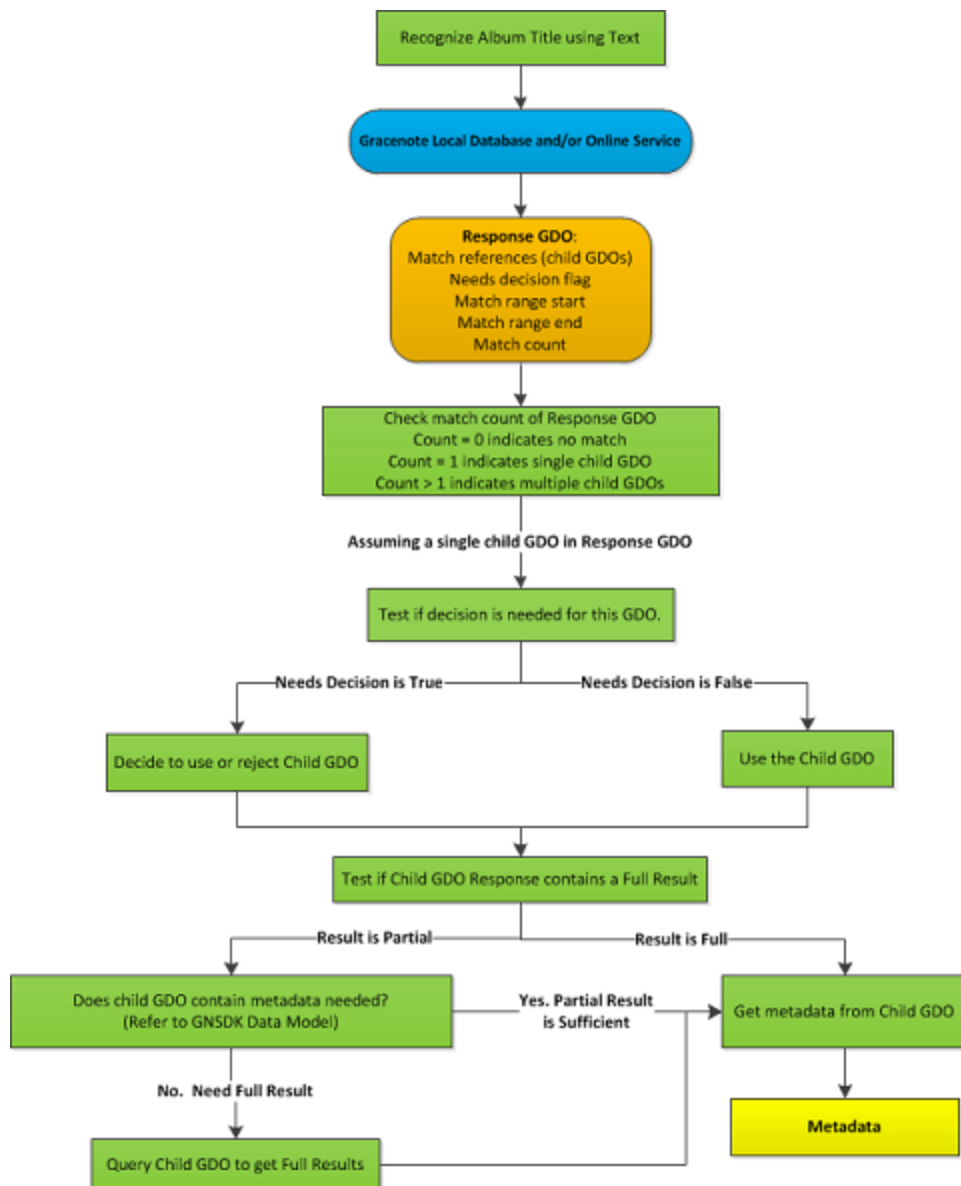
The application should also test the GDO to determine if it contains partial or full results. Use the GNSDK\_GDO\_VALUE\_FULL\_RESULT value key for this test. If the response is partial, the application can either use the partial information or perform an additional query to get the full results. In some cases, the information



returned in a partial response may be sufficient for the purpose of the query. If so, the application can simply get the values from the partial response.

For a list of values returned in a partial result, see the Data Model in the *GNSDK C API Reference*.

The following diagram shows the basic workflow, followed by the application steps in detail.



### GDO Workflow Steps for Album Title Text Lookup - Single GDO Response

1. Call `gnsdk_musicid_query_create()` to create a query handle.
2. Call `gnsdk_musicid_query_set_text()` with input text and input field `GNSDK_MUSICID_FIELD_ALBUM` to set the text query for an album title.

3. Call `gnsdk_musicid_query_find_albums()` to perform the query.
4. Test if Parent GDO response has multiple Child GDOs using `gnsdk_manager_gdo_value_get()` with `GNSDK_GDO_VALUE_RESPONSE_RESULT_COUNT`. For this example, assume a single GDO response was returned.
5. Test if a decision is needed for the Parent GDO using `gnsdk_manager_gdo_value_get()` with value key `GNSDK_GDO_VALUE_RESPONSE_NEEDS_DECISION`.
6. If decision is not needed, use the Child GDO. If a decision is needed, choose to use the GDO or reject it.
7. If using the GDO, call `gnsdk_manager_gdo_child_get()` with Child Key `GNSDK_GDO_CHILD_ALBUM` and its ordinal value of 1.
8. Test if the Child GDO Response contains a full result using `gnsdk_manager_gdo_value_get()` with Value Key `GNSDK_GDO_VALUE_FULL_RESULT`.
9. If the GDO contains a full result, jump to the last step. If the GDO contains partial results, decide if it contains the metadata needed for the query.
10. If partial result is sufficient, jump to the last step. If full results are needed, query the *Child GDO* to get the full results:
  - a. Set the handle to the *selected Child GDO* using `gnsdk_musicid_query_set_gdo()`.
  - b. Re-query using `gnsdk_musicid_query_find_albums()`.
11. Get metadata from Child GDO using `gnsdk_manager_gdo_value_get()` and value keys. See [Retrieving a Track GDO from an Album GDO](#)

### 3.10.6.2 GDO Workflow for Multiple GDO Matches

Gracenote identification queries often return multiple matches. For example, suppose that a Track exists on multiple Albums, such as the original Album, a compilation Album, and a greatest hits Album. In this case, the query returns a Response GDO that contains multiple child Album GDOs. Each GDO represents a possible Album match for the query. When this happens, the end-user needs to select which Album they want. Each Response GDO returns enough metadata for

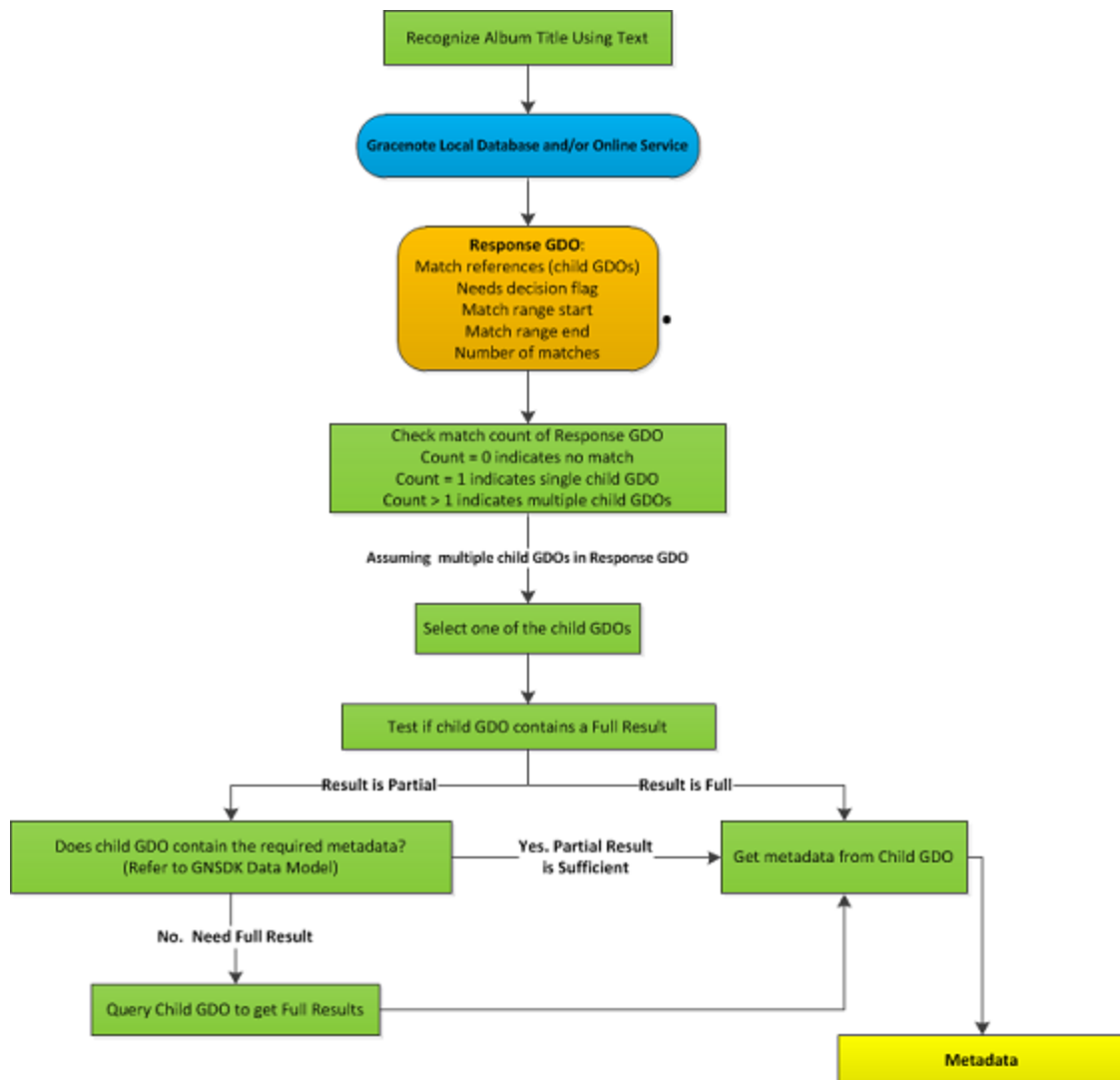
this purpose. Based on the user's selection, the application can then send another query to Gracenote to return the GDO for the chosen Album.

The diagram below shows the general application GDO workflow for multiple GDO responses. As described in [Matches that Require Decisions](#), *all* multiple GDO match responses require a decision by the application or end user. Therefore, it is optional to test if a multiple GDO match needs a decision.

In general, after choosing a Child GDO from the multiple matches, the application should test it to determine if it contains partial or full results. Use the GNSDK\_GDO\_VALUE\_FULL\_RESULT value key for this test. If the response is partial, the application can either use the partial information or perform an additional query to get the full results. In some cases, the information returned in a partial response may be sufficient for the purpose of the query. If so, the application can simply get the values from the partial response.

For a list of values returned in a partial result, see the Data Model documentation.

The following diagram shows the basic workflow, followed by the application steps in detail.



### GDO Workflow Steps for Album Title Text Lookup - Multiple GDO Response

1. Call `gnsdk_musicid_query_create()` to create a query handle.
2. Call `gnsdk_musicid_query_set_text()` with input text and input field `GNSDK_MUSICID_FIELD_ALBUM` to set the text query for an album title.
3. Call `gnsdk_musicid_query_find_albums()` to perform the query.

4. Test if Parent GDO response has multiple Child GDOs using `gnsdk_manager_gdo_value_get()` with `GNSDK_GDO_VALUE_RESPONSE_RESULT_COUNT`. For this example, assume a multiple GDO response was returned.
5. *Optional.* Test if a decision is needed for the Parent GDO using `gnsdk_manager_gdo_value_get()` with Value Key: `GNSDK_GDO_VALUE_RESPONSE_NEEDS_DECISION`. For multiple response GDOs, this always returns TRUE.
6. Choose a specific Child GDO using `gnsdk_manager_gdo_child_get()` with Child Key `GNSDK_GDO_CHILD_ALBUM` and its ordinal value (*1-based*).
7. Test if the Child GDO Response contains a full result using `gnsdk_manager_gdo_value_get()` with Value Key `GNSDK_GDO_VALUE_FULL_RESULT`.
8. If the GDO contains a full result, jump to the last step. If the GDO contains partial results, decide if it contains the metadata needed for the query.
9. If partial result is sufficient, jump to the last step. If full results are needed, query the Child GDO to get the full results:
  - a. Set the handle to the *selected Child GDO* using `gnsdk_musicid_query_set_gdo()`.
  - b. Re-query using `gnsdk_musicid_query_find_albums()`.
10. Get metadata from Child GDO using `gnsdk_manager_gdo_value_get()` and value keys. See [Retrieving a Track GDO from an Album GDO](#).

### 3.10.7 Common GDO Tasks

#### 3.10.7.1 Retrieving a Track GDO from an Album GDO

To retrieve a Track GDO from an Album GDO, use `gnsdk_manager_gdo_child_get()` and one of the following keys. For these keys, the term *track number* is the sequential number of the track on the album's CD jacket. The *ordinal* is the track position (1-based) in the array of tracks in an Album GDO.

- GNSDK\_GDO\_CHILD\_TRACK retrieves a Track GDO based on the *ordinal* position of the track.
- GNSDK\_GDO\_CHILD\_TRACK\_BY\_NUMBER retrieves a Track GDO that matches the provided *track number*.
- GNSDK\_GDO\_CHILD\_TRACK\_MATCHED returns one or more Track GDOs that were *matched* by the prior query.

In general, when retrieving tracks, follow these guidelines:

- Write code to manage the return of Partial album GDOs. Although this is likely a rare occurrence, do not assume that an Album GDO will contain all of the tracks of that album. If the Album returned is Partial, make a second query to retrieve the Full Album GDO.
- When retrieving matched tracks, always use the helper keys: GNSDK\_GDO\_CHILD\_TRACK\_MATCHED, and GNSDK\_GDO\_CHILD\_TRACK\_MATCHED\_NUM.

The following are examples of retrieving Tracks from an Album GDO. In this case, we assume the Album GDO is Full, containing all tracks of the album.

Assume a hypothetical TOC lookup of an album with 5 tracks. We get an album GDO that looks like:

```
<album>
  Ord 1    <Track 1>
  Ord 2    <Track 2>
  Ord 3    <Track 3>
  Ord 4    <Track 4>
  Ord 5    <Track 5>
</album>
```

In this case:

- GNSDK\_GDO\_CHILD\_TRACK with *ordinal* 4 returns Track 4.
- GNSDK\_GDO\_CHILD\_TRACK\_BY\_NUMBER with *track number* 4 returns Track 4.

The example below shows the Album GDO result after performing a match query, such as looking up a track by its fingerprint or title. Assuming this matched Track 4, we get an album GDO that looks like the following:

```
<album>
  Ord 1  <track_matched>4</track_matched>
  Ord 1  <Track 1>
  Ord 2  <Track 2>
  Ord 3  <Track 3>
  Ord 4  <Track 4>
  Ord 5  <Track 5>
</album>
```

This GDO contains two arrays:

- A track array: containing the actual tracks
- A `matched_track` array: containing track number of the matched track or tracks.

The `GNSDK_GDO_CHILD_TRACK*` keys operate on this GDO as follows:

- `GNSDK_GDO_CHILD_TRACK` with *ordinal* 4 returns Track 4.
- `GNSDK_GDO_CHILD_TRACK_BY_NUMBER` with *track number* 4 returns Track 4.
- `GNSDK_GDO_CHILD_TRACK_MATCHED` with *ordinal* 1 returns Track 4.

Also, `GNSDK_GDO_VALUE_TRACK_MATCHED_NUM` with *ordinal* 1 returns its value, the *number 4* which is the *track number* of the matched track.

### 3.10.7.2 Getting the Type for a GDO

If needed, your application can get the type of a GDO using the `gnsdk_manager_gdo_get_type()` function. For example, your application might need to determine a GDO's type after a text-based lookup:

```
gnsdk_manager_gdo_get_type(match_gdo, &gdo_type);

if (0 == strcmp(gdo_type, GNSDK_GDO_TYPE_ALBUM))
    /* Work with Album GDO */
```



```
else if (0 == strcmp(gdo_type, GNSDK_GDO_TYPE_CONTRIBUTOR))  
    /* Work with Contributor GDO */
```

### 3.10.7.3 Checking for Full and Partial Results

You can use the `GNSDK_GDO_VALUE_FULL_RESULT` value key to see if a child GDO contains full or partial metadata (`GNSDK_VALUE_FALSE` = partial, `GNSDK_VALUE_TRUE` = full).

**For example:**

```
...Perform album query and get response GDO  
  
/* Get first child album GDO from response GDO and check if it  
contains partial data */  
gnsdk_manager_gdo_child_get(response_gdo, GNSDK_GDO_CHILD_ALBUM, 1,  
&album_gdo);  
  
/* Is this a partial album? */  
gnsdk_manager_gdo_value_get(album_gdo, GNSDK_GDO_VALUE_FULL_RESULT, 1,  
&value_string);  
  
if (atoi(value_string) == GNSDK_VALUE_FALSE)  
{  
    printf("retrieving FULL RESULT\n");  
  
    /* Note that we are getting full results for the first child album  
GDO instead of  
    * presenting partial results from all child GDO matches to the end  
user to select from  
    */  
  
    /* Add child GDO back to the existing query handle to retrieve full  
result */  
    gnsdk_muscid_query_set_gdo(query_handle, album_gdo);  
  
    /* Query for this match in full*/  
    gnsdk_muscid_query_find_albums(query_handle, &response_gdo);  
  
    /* Get child GDO with full album results */  
    gnsdk_manager_gdo_child_get(response_gdo, GNSDK_GDO_CHILD_ALBUM, 1,  
&album_gdo);
```

...

#### 3.10.7.4 Checking Whether a Match Needs a Decision

You can use the `GNSDK_GDO_VALUE_RESPONSE_NEEDS_DECISION` boolean value key to see if a GDO needs a decision.

To resolve a decision, an application typically presents the end user with a subset of the matches' partial results (for example, album titles) and lets the user select the correct match or reject them all. Alternatively, the application may automatically select the first match or reject them all .



**Note:** In all cases, match results can be partial or full, so after selecting a match, the application should test it using `GNSDK_GDO_VALUE_FULL_RESULT`.

#### For example:

```
gnsdk_cstr_t value      = (gnsdk_cstr_t)GNSDK_NULL;
gnsdk_cstr_t needs_decision = 0;
gnsdk_uint32_t count     = 0;

/*
 * Get count and needs_decision values from response GDO
 */
gnsdk_manager_gdo_value_get(response_gdo, GNSDK_GDO_VALUE_RESPONSE_NEEDS_DECISION, 1, &value);
needs_decision = value;
gnsdk_manager_gdo_value_get(response_gdo, GNSDK_GDO_VALUE_RESPONSE_RESULT_COUNT, 1, &value);
count = atoi(value);

if (count > 0) // Either multiple matches or confidence in single
match is not high
{
    int user_selection = 1;

    if (0 == strcmp(needs_decision, GNSDK_VALUE_TRUE))
    (
        /*
```

```
    * Resolve match(es). Look at count to determine paging, depending
    on device.
    * Typically, you would use partial results from matches (for
    example, album titles)
    * and present them to end user to select from
    *
    * Set user_selection to user's choice or 0 if they do not make a
    selection
    */
    ...

}

if (user_selection)
{
    /* There is a single match */
    /* Test if GDO has full or partial metadata. If partial, get full
    (if desired) */

    ...

}
}
else
{
    /* No match returned */
}
}
```

### 3.10.7.5 Serializing a GDO

You can serialize a GDO to save it for later use in an application. Serializing a GDO retains only key information needed for common GDO functions. Other information is discarded.

To restore the discarded information, your application must request the GDO again. An application can reconstitute (de-serialize) a serialized GDO and use it for subsequent processing.

For a list of values returned in a partial result, see the Data Model in the *GNSDK C API Reference*.

**Important:** A serialized GDO is not a static identifier. You cannot use it as a comparison identifier for any purposes, including straight comparisons, caching, indexing, and so on.

### 3.10.7.6 Rendering a GDO as XML

GNSDK supports rendering GDOs as XML. This is an advanced feature, but may be useful to track user choices, retain query history, or provide data to other applications. Rendering to XML is also helpful when testing queries to see their results. For more information, see *"Rendering a GDO as XML" on page 163*.

### 3.10.8 GDO Navigation Examples

Gracenote Data Objects (GDOs) are the primary identifiers used to access Gracenote metadata.

#### 3.10.8.1 Example: Looking Up an Album by a TOC

Sample Application: musicid\_lookup\_album\_toc/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and MusicID module, and initialize SQLite and local lookup
2. Enable logging, load a locale, and get a User handle
3. Perform a MusicID query based on TOC identifier
4. Get album child GDOs from album response GDO
5. Access and display album titles
6. Release resources and shutdown GNSDK and MusicID module

**Sample output:**

```
GNSDK Product Version   : 3.05.0.721    (built 2013-04-02 22:29-0700)

*****MusicID TOC Query*****
  Match count: 8
    Title: Come Away With Me
```

```
Title: è¸œµ°é«œłž
Title: Come Away With Me
Title: Come Away With Me
Title: Feels Like Home
Title: Norah Jones: Come Away With Me
Title: Come Away With Me
Title: Feels Like Home
Final album:
Title: Come Away With Me
```

### 3.10.8.2 Example: Accessing Album and Track Metadata Using Album GDO

This example uses an Album GDO to access Album metadata: artist, credits, title, year, and genre, as well as basic Track metadata: artist, credits, title, track number, and genre.

Sample Application: `musicid_gdo_navigation/main.c`

Application steps:

1. Authenticate caller, initialize Manager and MusicID module, and initialize SQLite and local lookup.
2. Enable logging, load a locale, and get a User handle.
3. Perform a MusicID query based on TUI identifier.
4. Get album child GDO from album response GDO.
5. Access and display album metadata.
6. Release resources and shutdown Manager and the MusicID module.

Sample output:

```
GNSDK Product Version : 3.05.0.721 (built 2013-04-02 22:29-0700)

*****Sample MusicID Query*****
Match.

***Navigating Result GDO***
Album:
  Package Language: English
  Credit:
    Contributor:
      Name Official:
```

```
    Display: Nelly
    Origin Level 1: North America
    Origin Level 2: United States
    Origin Level 3: Missouri
    Origin Level 4: St. Louis
    Era Level 1: 2000's
    Era Level 2: 2000's
    Era Level 3: 2000's
    Artist Type Level 1: Male
    Artist Type Level 2: Male
Title Official:
    Display: Nellyville
Year: 2002
Genre Level 1: Urban
Genre Level 2: Western Hip-Hop/Rap
Genre Level 3: Midwestern Rap
Album Label: Universal
Total In Set: 1
Disc In Set: 1
Track Count: 19
Track:
    Track TUI: 30716058
    Track Number: 1
    Title Official:
        Display: Nellyville
Track:
    Track TUI: 30716059
    Track Number: 2
    Credit:
        Contributor:
            Name Official:
                Display: Nelly Feat. Cedric The Entertainer & La La
    Title Official:
        Display: Gettin It Started
Track:
    Track TUI: 30716060
    Track Number: 3
    Title Official:
        Display: Hot In Herre
    Year: 2002
Track:
    Track TUI: 30716061
    Track Number: 4
    Title Official:
        Display: Dem Boyz
```

```
Track:
  Track TUI: 30716062
  Track Number: 5
  Title Official:
    Display: Oh Nelly
Track:
  Track TUI: 30716063
  Track Number: 6
  Title Official:
    Display: Pimp Juice
Track:
  Track TUI: 30716064
  Track Number: 7
  Title Official:
    Display: Air Force Ones
  Year: 2002
Track:
  Track TUI: 30716065
  Track Number: 8
  Credit:
    Contributor:
      Name Official:
        Display: Nelly Feat. Cedric The Entertainer & La La
  Title Official:
    Display: In The Store
Track:
  Track TUI: 30716066
  Track Number: 9
  Credit:
    Contributor:
      Name Official:
        Display: Nelly Feat. King Jacob
      Origin Level 1: North America
      Origin Level 2: United States
      Origin Level 3: Missouri
      Origin Level 4: St. Louis
      Era Level 1: 2000's
      Era Level 2: Early 2000's
      Era Level 3: Early 2000's
      Artist Type Level 1: Male
      Artist Type Level 2: Male Duo
  Title Official:
    Display: On The Grind

... More Tracks
```

... More Matches

### 3.10.9 Working with Non-GDO Identifiers

In addition to GDOs, Gracenote supports other media element identifiers, as shown in the following table. All of these represent TUI/Tag pairs.

Identifier	Description	Best Use
TUI	TUI stands for "Title Unique Id", and is the core identifier for Gracenote. This is a value Gracenote assigns to every object in the system. This id is unique for every object, but as Gracenote can have multiple objects for the same data (eg: multiple album records for the same album), the TUI can not be used for comparison between objects. In other words, just because two TUIs don't match does not conclusively mean that the data they refer to is different.  The TUI must be used in conjunction with the TUITag (see below).	The TUI is useful for storing a value of a record for retrieval of the same record at a later date. It can also be used as a unique value for a record as it will not be repeated by other Gracenote objects.
TUITag	The TUITag is generated by hashing the TUI. This id is used in conjunction with TUI values when querying a TUI. The TUITag prevents applications from iterating all possible TUI values, retrieving all Gracenote records. Without a valid TuiTag the record for a TUI cannot be retrieved.	The TuiTag must be provided with the TUI values whenever the record for a TUI is looked up from Service.
TagID/ProductID	The TagID is an obfuscated combination of the TUI and TUITag values. This ID is designed to be stored in the 'Tag' data of various media files that make use of the metadata from the Gracenote SDK. (for example, the TagID should be stored in the ID3v2 tag for MP3 files when using Gracenote metadata to create the MP3 file). ProductID is another name for the TagID, and represents the same data.	Use as the value to store in media file tags. You can also use it as a way to have a single value represent the TUI and Tag, although the GnID or GnUID (described below) are better for this.



Identifier	Description	Best Use
GnID	The GnID is a non-obfuscated combination of the TUI and TUITag values (it is of the form <TUI>-<TUITag>). This is just a convenient 'single value' form of the TUI and TUITag values since they are generally used together.	Use the GnID to refer to a specific record when you may want to refetch the exact record at a future time. This ID works well with the WebAPI for Gracenote and can be used as an interchange between GNSDK and WebAPI.
GnUID	The GnUID is a condensed version of the TUI and TUITag combination. This is the latest incarnation of a representation for the TUI and TUITag pair of values. The GnUID is much smaller than any of the other values, while still allowing for queries for the record it represents from the Gracenote Service. Further, the GnUID has type information within it which allows GNSDK to know what object it represents. This in turn can gain certain efficiencies in GNSDK when using this ID.	The GnUID is the best value to refer to a specific record when you may want to refetch the exact record at a future time. There is limited support for this value in the Gracenote WebAPI or with legacy versions of GNSDK (2.x and older).
GlobalID	<p>The GlobalID represents the Album Title or the Artist Name of an album record. The GlobalID is the same value for albums of the same title, or for artists that are equivalent (not just the same name).</p> <p>Functionality depends on integration with the Nuance VoCon Music Premium SDK.</p>	The GlobalID is used as an interchange between GNSDK and the Nuance VoCon Music Premium SDK. The Nuance SDK will use the GlobalID to quickly fetch the correct transcriptions for this value. If you are not using Nuance, you can ignore this value.

### 3.10.9.1 Examples of Non-GDO Identifiers

The following table shows examples of non-GDO identifiers.

Identifier	Example
TUI/Tag	12345567/ABCDEF1234567890ABCDEF1234567890
GNID	12345567-ABCDEF1234567890ABCDEF1234567890
GNUID	ABCD1234567890
TAGID and ProductID	DATA12345567FOOABCDEF1234567890ABCDEF1234567890DATA

### 3.10.9.2 GNSDK Identifiers Comparison Matrix

The following table lists the strengths and weaknesses of the different Gracenote identifiers.

Identifier	Pros	Cons
Serialized GDO	<p>Good for getting back to the original metadata.</p> <p>Good for joining responses and requests between different Gracenote products</p> <p>Does not require knowledge of the object type</p>	<p>Cannot be used for comparison to other serialized GDO values</p> <p>Not useful for integrating GNSDK to other metadata sources, as only GNSDK supports GDOs</p> <p>Relatively large and requires much storage space</p>
TUI /TUI Tag Pair	<p>Good for comparison</p> <p>Good for integrating GNSDK with other Gracenote metadata sources</p>	<p>Cannot be used for getting back to the original metadata, due to the lack of a Tag ID to perform the query</p> <p>Cannot be efficiently used for joining different Gracenote products, as it does not contain enough metadata.</p> <p>Requires knowledge of the object type</p> <p>Requires an accompanying tag, as it is not a single input</p>

Identifier	Pros	Cons
TagID (ProductID)	<p>Good for comparison</p> <p>Good for getting back to original metadata.</p> <p>Concise and does not require large amounts of storage space</p> <p>Single input and does not require an accompanying tag</p>	<p>Cannot be used among different Gracenote products as a Product ID/Tag ID</p> <p>contains only enough information for certain SDKs</p> <p>Cannot be efficiently used for integrating the GNSDK to other metadata sources, as Product IDs/Tag IDs are not generally supported (however, they can be deconstructed to a TUI )</p> <p>Requires knowledge of the object type</p>

### 3.10.9.3 Converting Non-GDO Identifiers to GDOs

This section shows examples that create GDO identifiers from Non-GDO identifiers.

#### Example: Creating a GDO from an Album TUI

```
gnsdk_gdo_handle_t query_gdo;
gnsdk_cstr_t tui = "12345678";
gnsdk_cstr_t tui_tag = "ABCDEF1234567890ABCDEF1234567890";
error = gnsdk_manager_gdo_create_from_id(tui, tui_tag, GNSDK_ID_
SOURCE_ALBUM, &query_gdo);
if (!error)
error = gnsdk_musicid_query_set_gdo(query_gdo);
```

#### Example: Creating a GDO from a Track TagID

```
gnsdk_gdo_handle_t query_gdo;
gnsdk_cstr_t tagid =
"3CD3N43R15145217U37894058D7FCB938ADFA97874409F9531B2P3";
error = gnsdk_manager_gdo_create_from_id(tagid, GNSDK_NULL, GNSDK_ID_
SOURCE_TRACK, &query_gdo);
if (!error)
error = gnsdk_musicid_query_set_gdo(query_gdo);
```

### Example: Creating a GDO from an Video Product ID

```
gnsdk_gdo_handle_t query_gdo;
gnsdk_cstr_t productid =
"3CD3N43R15145217U37894058D7FCB938ADFA97874409F9531B2P3";
error = gnsdk_manager_gdo_create_from_id(productid, GNSDK_NULL, GNSDK_ID_
SOURCE_VIDEO_PRODUCT, &query_gdo);
if (!error)
error = gnsdk_video_query_set_gdo(query_gdo);
```

### Example: Creating a GDO from a CDDBDID

```
gnsdk_gdo_handle_t query_gdo;
gnsdk_cstr_t cddbaid = "4+029C475EFCF8D469C78A644DEBFABF91+5795407";
error = gnsdk_manager_gdo_create_from_id(cddbaid, GNSDK_NULL, GNSDK_ID_
SOURCE_CDDBDID, &query_gdo);
if (!error)
error = gnsdk_musiciid_query_set_gdo(query_gdo);
```

### Example: Creating a GDO from Raw CDDBDID

A Raw CDDBDID is a decomposition of the CDDBDID above.

```
gnsdk_gdo_handle_t query_gdo;
gnsdk_cstr_t mui = "5795407";
gnsdk_cstr_t mediaid = "029C475EFCF8D469C78A644DEBFABF91";
error = gnsdk_manager_gdo_create_from_id(mui, mediaid, GNSDK_ID_
SOURCE_CDDBDID, &query_gdo);
if (!error)
error = gnsdk_musiciid_query_set_gdo(query_gdo);
```

### **3.10.10** *Rendering a GDO as XML*

To present GDO metadata to a user, or to process its contents for other uses, an application can render it in XML format.

You can do this with the following call:

```
/*
 * Render the Album GDO as XML to the "xml" parameter
```

```
*/
gnsdk_manager_gdo_render(album_gdo, GNSDK_GDO_RENDER_XML_FULL, &xml);
```

The following options are available when rendering to XML

Option	Meaning
GNSDK_GDO_RENDER_XML_MINIMAL	Renders minimal metadata.
GNSDK_GDO_RENDER_XML_STANDARD	Renders Standard metadata.
GNSDK_GDO_RENDER_XML_CREDITS	Renders any credit metadata.
GNSDK_GDO_RENDER_XML_GDOS	Renders any serialized GDO values.
GNSDK_GDO_RENDER_XML_IDS	Renders any Gracenote product ID values.
GNSDK_GDO_RENDER_XML_SEEDS	Renders any Gracenote Discover Seed values.
GNSDK_GDO_RENDER_XML_RAW_TUIS	Renders any Gracenote TUI values.
GNSDK_GDO_RENDER_XML_SUBMIT	Renders any Gracenote data supported for Submit editable GDOs. This rendered data includes both the supported editable and non-editable data.
GNSDK_GDO_RENDER_XML_GENRE	Renders any Gracenote genre values.

Option	Meaning
GNSDK_GDO_RENDER_XML_GENRE_META	Renders any Gracenote meta-genre (coarser) values.
GNSDK_GDO_RENDER_XML_GENRE_MICRO	Renders any Gracenote micro-genre (finer) values.
GNSDK_GDO_RENDER_XML_DEFAULT	Renders the default metadata. Identifiers are not included. Equivalent to GNSDK_GDO_RENDER_XML_STANDARD   GNSDK_GDO_RENDER_XML_GENRE.
GNSDK_GDO_RENDER_XML_FULL	Renders the majority of metadata. Identifiers not included. Equivalent to GNSDK_GDO_RENDER_XML_DEFAULT   GNSDK_GDO_RENDER_XML_CREDITS

The values in the above table can be OR'd together to form combinations of data as needed. For example, you can use **GNSDK\_GDO\_RENDER\_XML\_SEEDS | GNSDK\_GDO\_RENDER\_XML\_FULL** as a parameter.

#### 3.10.10.1 Example: Rendering a GDO as XML

Sample application:submit\_album/main.c

This example submits a new album to Gracenote, and contains code that renders a GDO to XML using GNSDK\_GDO\_RENDER\_XML\_SUBMIT.

#### 3.10.11 Creating a GDO From XML

You may reverse the process described above by calling `gnsdk_manager_gdo_create_from_render()`. Pass in the an XML string, and the function returns back a handle to the created GDO.

### 3.11 Identifying Music

GNSDK supports identification of both non-streaming and streaming music.

Non-streaming music generally refers to music stored as a file or on a CD. You can identify non-streaming music using a CD TOC, a Gracenote identifier, or other information extracted from an audio file.

Streaming music refers to music that is delivered in real-time as an end user listens. Listening to a song on the radio or playing a song from a media player are both examples of streaming music. You can identify streaming music using audio fingerprints generated from a streaming audio source.

### **3.11.1 Identifying Music Using a CD TOC**

You can use MusicID-CD to identify an audio CD TOC. For more information, see *"CD TOC Recognition" on page 11*.

#### **3.11.1.1 Example: Identifying an Album Using a CD TOC**

The example 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:

```
gnsdk_storage_sqlite_initialize(sdkmgr_handle);  
gnsdk_lookup_local_initialize(sdkmgr_handle);
```

You must also set the query option to do a local lookup:

```
gnsdk_musicid_query_option_set(  
    query_handle,  
    GNSDK_MUSICID_OPTION_USE_LOOKUP_LOCAL,  
    "true"  
);
```

Other than these additions, setting up queries and processing results works in the same way for both local and online lookups.

Sample Application: musicid\_lookup\_album\_toc/main.c

Application Steps:

1. Authenticate caller and initialize GNSDK and MusicID modules.
2. Initialize SQLite, local lookup, logging, load a locale, and get a User handle
3. Perform album query using CD TOC.
4. Access and display metadata from matches and pick one.
5. Release resources and shutdown GNSDK and MusicID modules.

**Sample output:**

```
GNSDK Product Version : 3.05.0.721 (built 2013-04-02 22:29-0700)

*****MusicID TOC Query*****
  Match count: 8
    Title: Come Away With Me
    Title: è;œèµ°é«œłž
    Title: Come Away With Me
    Title: Come Away With Me
    Title: Feels Like Home
    Title: Norah Jones: Come Away With Me
    Title: Come Away With Me
    Title: Feels Like Home
  Final album:
    Title: Come Away With Me
```

### **3.11.2** *Identifying Music Using Text*

#### **3.11.2.1** *Creating and Executing a Query for a Text-based Lookup*

The first step in performing a text-based lookup is to create a music query, which returns a query handle that you will use in subsequent calls:

```
gnsdk_musicid_query_create( user_handle, GNSDK_NULL, GNSDK_NULL,
&query_handle );
```

The next step in creating the query is to set the input text fields, based on the information you have available. The possible input fields are:

- GNSDK\_MUSICID\_FIELD\_ALBUM (album title)
- GNSDK\_MUSICID\_FIELD\_TITLE (track title)



- GNSDK\_MUSICID\_FIELD\_ALBUM\_ARTIST (album artist)
- GNSDK\_MUSICID\_FIELD\_TRACK\_ARTIST (track artist)
- GNSDK\_MUSICID\_FIELD\_COMPOSER (track composer; only supported for classical music)

Call the `gnsdk_musicid_query_set_text()` function to set each input field. For example, the following call sets the album title "Dark Side of the Moon" to be used as an input field:

```
gnsdk_musicid_query_set_text( query_handle, GNSDK_MUSICID_FIELD_ALBUM,
"Dark Side of the Moon" );
```



**Note:** If both TRACK\_ARTIST and ALBUM\_ARTIST are provided and are different, TRACK\_ARTIST is given preference in the search.

Finally, to execute the query, call:

```
gnsdk_musicid_query_find_matches( query_handle, &response_gdo );
```

### 3.11.2.2 Processing Text-based Lookup Results

After executing the text-based query, you need to determine which “best-fit” objects were returned. To do this, iterate through the objects, and get the match GDO and then the GDO type, using the following functions:

```
gnsdk_manager_gdo_child_get( response_gdo, GNSDK_GDO_CHILD_MATCH, ord,
&match_gdo );
```

```
gnsdk_manager_gdo_get_type( match_gdo, &gdo_type );
```

The GDO type will be one of the following types:

- GNSDK\_GDO\_TYPE\_ALBUM
- GNSDK\_GDO\_TYPE\_CONTRIBUTOR

Compare the GDO type to these types, as shown in the following example:

```
if (0 == strcmp(gdo_type, GNSDK_GDO_TYPE_ALBUM))
{
    printf( "Album match\n" );
    // Display album information.
}
else if (0 == strcmp(gdo_type, GNSDK_GDO_TYPE_CONTRIBUTOR))
{
    printf( "Contributor match\n" );
    // Display contributor information.
}
```



**Note:** If an album GDO is returned and you need to make a follow-up query, use `gnsdk_musicid_query_find_albums()`. If the result is a contributor GDO, no follow-up query is needed.

For information about working with collaborative artist information for contributors, see *"Accessing Collaborative Artists Metadata" on page 218*.

For more information about navigating through the GDO type hierarchy, see *"About Gracenote Data Objects (GDOs)" on page 138*.

Text searches can return multiple matches containing partial results. For information on handling this, see *"Full and Partial Metadata Results" on page 140*

### 3.11.2.3 Example: Text Lookup for a Track

This example performs a sample text query with album, track and artist inputs.

Sample Application: `musicid_lookup_matches_text/main.c`

Application Steps:

1. Authenticate caller, initialize GNSDK Manager, enable logging, and initialize SQLite.
2. Initialize MusicID, load a Locale, and get a User handle.
3. Perform a text query with album, track and artist inputs
4. Perform a text query with just an album name
5. Perform a text query with just an artist name

6. Perform a text query with track title and artist name
7. Release resources and shutdown GNSDK and MusicID module.

**Sample output:**

```
GNSDK Product Version   : 3.05.0.721    (built 2013-04-02 22:29-0700)

*****MusicID Text Match Query*****
album title   : Supernatural
track title   : Africa Bamba
artist name   : Santana
  Match count: 10
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
    Title: Supernatural
  Final match:
    Title: Supernatural

*****MusicID Text Match Query*****
album title   : çœæ^`72â
  Match count: 10
    Title: çœæ^`72è©Š
    Title: çœæ^`72â
    Title: çœæ^`72â
    Title: çœæ^`72è©Š
    Title: çœæ^`72è©Š
    Title: çœæ^`72â
    Title: çœæ^`72è©Š
    Title: çœæ^`72â
    Title: çœæ^`72â
    Title: çœæ^`72â
  Final match:
    Title: çœæ^`72è©Š

*****MusicID Text Match Query*****
artist name   : Philip Glass
  Match count: 10
    Title: The Hours
```

```
Title: Glass: Solo Piano
Title: Mishima
Title: The Illusionist
Title: Glass: Etudes For Piano, Vol. 1/1-10
Title: Book Of Longing [Disc 1]
Title: Notes on a Scandal
Title: Philip Glass: The Fog of War
Title: Oeuvres Majeures [Disc 1]
Title: North Star
Final match:
  Title: The Hours

*****MusicID Text Match Query*****
artist name  : Bob Marley
  Match count: 10
    Title: One Love
    Title: Songs Of Freedom [Disc 2]
    Title: Natural Mystic
    Title: Songs Of Freedom [Disc 4]
    Title: Songs Of Freedom [Disc 1]
    Title: Songs Of Freedom [Disc 3]
    Title: Chant Down Babylon
    Title: Exodus
    Title: The Very Best In Memoriam
    Title: Survival
  Final match:
    Title: One Love

*****MusicID Text Match Query*****
track title  : Purple Stain
artist name  : Red Hot Chili Peppers
  Match count: 10
    Title: Californication
    Title: Live In Hyde Park [Disc 2]
    Title: Californication
    Title: Live At Slane Castle
    Title: Live At Slane Castle
    Title: Californication
    Title: Californication
    Title: Californication
    Title: Californication
    Title: Live At Slane Castle
  Final match:
    Title: Californication
```

```

*****MusicID Text Match Query*****
track title   : Eyeless
artist name   : Slipknot
  Match count: 10
    Title: Slipknot
    Title: Slipknot
    Title: Slipknot
    Title: Slipknot [Bonus Tracks]
    Title: Slipknot
    Title: Slipknot: 10th Anniversary Edition
    Title: Slipknot
    Title: 742617000027
    Title: "Left Behind" Greatest Hits
    Title: Slipknot
  Final match:
    Title: Slipknot

```

### 3.11.2.4 Best Practices for MusicID Text Match Queries

You can use the `GnMusicId` class' `FindAlbums` and `FindMatches` methods to identify music files in a collection based on one or more text values. This kind of query is called a *text match query*.

Possible text inputs are artist name, album title, and track title. You can specify these as fields in the query handle. The more fields you provide, the better and more accurate the result, so you should provide as many as possible.

The query response will contain either albums (with or without matched track data), or contributors (artists) if an album match was not available.



**Note:** Currently, online queries currently do not return contributors. Only albums are returned.

The following table shows the usable metadata returned for a given set of inputs.

Input			Use This Data
Album Title	Contributor/Artist Name	Track Title	
✓	✓	✓	Use all information: <ul style="list-style-type: none"> <li>Album</li> <li>Album's primary artist</li> <li>Matched track if available</li> <li>Matched track's artist if available</li> </ul>
✓	✓	✗	Use all available information: <ul style="list-style-type: none"> <li>Album</li> <li>Album's primary artist</li> </ul>
✓	✗	✓	Use all available information: <ul style="list-style-type: none"> <li>Album</li> <li>Album's primary artist</li> <li>Matched track if available</li> <li>Matched track's artist if available</li> </ul>
✗	✓	✓	Use all available information: <ul style="list-style-type: none"> <li>Album</li> <li>Album's primary artist</li> <li>Matched track if available</li> <li>Matched track's artist if available</li> </ul>
✗	✓	✗	<ul style="list-style-type: none"> <li>If the response is an album, only use the album's primary artist.</li> <li>If the response is an artist, use all available information.</li> </ul>
✓	✗	✗	Do not send this query.
✗	✗	✓	Do not send this query.

### 3.11.2.5 Identifying Music Using Batch Processing

Batch processing allows multiple queries to be performed in a single request to Gracenote Service. A single batch request can contain different query types, and each query can have different inputs. For example, a batch lookup can have a query for finding matches with text inputs in addition to finding matches based on fingerprint input.

#### Creating a Batch Query

You create queries the same way as any other MusicID query, but instead of invoking a find function on a single query, you add the query to a batch lookup handle, and then call a batch execute function. Each query requires a unique string identifier. The identifier allows the application to associate the query with its results. Creating a batch query with an identifier already in use results in an error. Gracenote recommends batches between 20 and 50 queries. In creating your batch queries please note the following:

- Batches containing 20 queries offer a measurable reduction in average query processing time as well as uplink and downlink transmission size without much of an increase to peak memory usage.
- Batches containing approximately 50 queries offers some reduction in average query processing time as well as uplink and downlink data transmissions, but the peak memory usage is higher.
- Batches greater than 50 queries often increases average query processing time as well as uplink and downlink data transmissions and require larger peak memory usage.
- Batch sizes of one are discouraged. It is more efficient to use a standard query if only one result is required.

The following example creates an array of 5 text queries, and then calls the batch function `gnsdk_musicid_batch_query_create()` to create the query, and `gnsdk_musicid_batch_query_set_text()` to set query text:

```
/*
 NOTE: Return code checking has been removed for brevity.  Always
 check GNSDK return codes.
 Create the batch query handle.
*/
gnsdk_musid_batch_create(
    user_handle,      /* From program initialization */
    GNSDK_NULL,      /* User callback function */
    GNSDK_NULL,      /* Optional data to be passed to the callback */
    &batch_handle     /* Batch handle returned */
);

/*
 Add queries to the batch query handle. This section of code adds
 queries based on the batch handle,
 and within that handle, on the unique name of the individual query.
*/
/* create individual queries for the batch handle, each query must
 have a unique identifier */
/* Track 1 */
gnsdk_musid_batch_query_create(batch_handle, "track_1");
gnsdk_musid_batch_query_set_text(batch_handle, "track_1", GNSDK_
MUSICID_FIELD_TRACK_ARTIST, "Jesse Cook");
gnsdk_musid_batch_query_set_text(batch_handle, "track_1", GNSDK_
MUSICID_FIELD_ALBUM, "Ultimate Jesse Cook (Disc 2)");
gnsdk_musid_batch_query_set_text(batch_handle, "track_1", GNSDK_
MUSICID_FIELD_TITLE, "On Walks The Night");

/* Track 2 */
gnsdk_musid_batch_query_create(batch_handle, "track_2");
gnsdk_musid_batch_query_set_text(batch_handle, "track_2", GNSDK_
MUSICID_FIELD_TRACK_ARTIST, "The Cure");
gnsdk_musid_batch_query_set_text(batch_handle, "track_2", GNSDK_
MUSICID_FIELD_ALBUM, "Join The Dots: B-Sides and Rarities 1978-2001
(Box Set CD 1)");
gnsdk_musid_batch_query_set_text(batch_handle, "track_2", GNSDK_
MUSICID_FIELD_TITLE, "A Man Inside My Mouth");

/* additional queries can be created based on other inputs such as CD
TOC, fingerprint and identifier.*/
```



## Performing a Batch Query and Retrieving Data

After the batch query is set, you retrieve the results from all queries with one call to `gnsdk_muscid_batch_find_matches()` or `gnsdk_muscid_batch_find_albums()`. The individual queries are executed in one request to Gracenote Service. You retrieve the responses from the batch handle using `gnsdk_muscid_batch_response_get()`, based on the unique identifier for the query. Queries executed in batch are limited to a single response. Because of this, `GNSDK_MUSCID_OPTION_RESULT_SINGLE` is automatically set and cannot be unset. The following code executes the batch query based on the queries in the previous snippet, and retrieves the individual response GDOs: It then releases the batch query handle:

```
/* Perform batch query using the batch query handle. */
gnsdk_muscid_batch_find_matches(batch_handle);

/*
   At this point all responses have been created with one call to the
   database.  You can now retrieve response GDO data
   based on the unique query id.
*/

/* Track 1 */
gnsdk_muscid_batch_response_get(batch_handle, "track_1", &response_
gdo);
/* process Track 1 response */

/* Track 2 */
gnsdk_muscid_batch_response_get(batch_handle, "track_2", &response_
gdo);
/* process Track 1 response */

/*
   After you have completed processing, release any query handles or
   GDOs as required.
   Following completion of batch queries, you must also release the
   batch query handle.
*/
gnsdk_muscid_batch_release(batch_handle);
```

## Setting Batch Lookup Options

Options set on a batch lookup affect all queries added to the batch lookup handle. Options are the same for batch queries as for non-batched queries. The following code sets the lookup mode for the batch queries to online nocache mode.

```
/* Set option for online lookup */
gnsdk_musicid_batch_option_set(batch_handle, GNSDK_MUSICID_OPTION_
LOOKUP_MODE, GNSDK_LOOKUP_MODE_ONLINE_NOCACHE);
```

## Examining Batch Functions

The following table contains the listing of batch query functions available. Consult `gnsdk_musicid.h` for further information.

Function Name	Description
<code>gnsdk_musicid_batch_clear()</code>	Removes all individual queries from a MusicID batch query.
<code>gnsdk_musicid_batch_create()</code>	Creates a MusicID batch query handle.
<code>gnsdk_musicid_batch_find_albums()</code>	Performs a MusicID batch query for album results.
<code>gnsdk_musicid_batch_find_matches()</code>	Performs a MusicID batch query for matches of the best fit type (albums or contributors).
<code>gnsdk_musicid_batch_option_get()</code>	Retrieves an option for a given MusicID batch query handle.
<code>gnsdk_musicid_batch_option_set()</code>	Sets an option for a given MusicID batch query handle. Options set on the batch handle apply to individual queries processed in the batch.
<code>gnsdk_musicid_batch_query_add_toc_offset()</code>	Sets CD TOC offset values for an individual query.

Function Name	Description
<code>gnsdk_musicid_batch_query_create()</code>	Creates a new individual query to be processed as part of the MusicID batch query.
<code>gnsdk_musicid_batch_query_set_fp_data()</code>	Sets externally-generated fingerprint data for an individual query.
<code>gnsdk_musicid_batch_query_set_gdo()</code>	Sets GDO input for an individual query.
<code>gnsdk_musicid_batch_query_set_text()</code>	Sets text inputs for an individual query.
<code>gnsdk_musicid_batch_query_set_toc_string()</code>	Sets CD TOC input for an individual query.
<code>gnsdk_musicid_batch_release()</code>	Invalidates and releases resources for a given MusicID batch query handle.
<code>gnsdk_musicid_batch_response_get()</code>	Gets an individual query's response from a batch query using it's unique identifier.
<code>gnsdk_musicid_batch_set_locale()</code>	Sets a locale to be used as input for all queries added to a MusicID batch query.

### Example: Executing Batch Lookups

Sample Application: `musicid_lookup_batch_matches_text/main.c`

Application Steps:

1. Authenticate User and initialize Manager and other modules as necessary.
2. Create two batch query arrays.
3. Perform the batch queries and trace through response GDOs for results.
4. Release resources and shutdown GNSDK and modules.

### 3.11.3 Using MusicID-File

As described in [MusicID-File Overview](#), MusicID-File provides three processing techniques to identify audio files: TrackID, AlbumID, and LibraryID. This topic describes the basic steps your application should follow to implement to use these processes:

#### 3.11.3.1 TrackID and AlbumID Implementation Steps

1. Initialize the GNSDK and the MusicID-File and DSP (for fingerprinting) modules
2. Create a MusicID-File query handle
3. Create a FileInfo object for each media file you want to process, and add it to MusicID-File query handle.
4. Add identification (fingerprinting and metadata) to each FileInfo object.
5. Set query options.
6. Make TrackID or AlbumID query.
7. Get query results, See [Getting Results with TrackID and AlbumID](#).
8. Release resources and shutdown MusicID-File, DSP, and GNSDK.

#### 3.11.3.2 LibraryID Implementation Steps

1. Initialize the GNSDK and the MusicID-File and DSP (for fingerprinting) modules.
2. Author callback functions to handle fingerprinting, metadata, returned statuses, returned results, and so on..
3. Create a MusicID-File query handle with callbacks array.
4. Create a FileInfo object for each media file you want to process, and add it to the MusicID-File query handle.
5. Set query options.
6. Make LibraryID query.

7. In callbacks, add identification (fingerprinting and metadata) to each FileInfo object.
8. In callbacks, get query results. See [Getting Results with LibraryID](#)
9. Release resources and shutdown MusicID-File, DSP, and GNSDK.

### 3.11.3.3 Initialization

Your application needs to have the following include, which by default includes all GNSDK modules:

```
#include "gnsdk.h" /* Includes all modules */
```

To use MusicID-File and DSP, you need to make the following calls after initializing the GNSDK and getting a Manager handle:

```
gnsdk_musicidfile_initialize(sdkmgr_handle);  
gnsdk_dsp_initialize(sdkmgr_handle); /* Required for fingerprinting  
*/
```

### 3.11.3.4 Creating a MusicID-File Query Handle

After standard application initialization, the next step is to create a MusicID-File query handle:

#### TrackID and AlbumID

```
/* Create the MusicID-File query handle */  
gnsdk_musicidfile_query_create(  
    user_handle,  
    GNSDK_NULL, /* Callback function for status and progress */  
    GNSDK_NULL, /* Callback data */  
    &query_handle  
);
```

#### LibraryID

LibraryID requires you to create the query handle with the callbacks that will be used for all processing:

```

/* Create array of callbacks */
gnsdk_musicidfile_callbacks_t midf_callbacks = {
    _musicidfile_status_callback, /* Status callback */
    _get_fingerprint_callback,    /* Fingerprinting callback */
    _get_metadata_callback,       /* Set metadata callback */
    _result_available_callback,    /* Results callback */
    GNSDK_NULL,                  /* Results not found callback */
    GNSDK_NULL,                  /* MusicID complete callback */
};

/* Create the MusicID-File handle */
gnsdk_musicidfile_query_create(
    user_handle,
    &midf_callbacks, /* Array of callbacks */
    GNSDK_NULL,     /* Callback user data */
    &query_handle    /* Query handle */
);

```

### 3.11.3.5 Creating FileInfo Objects

For each Track media file you want to identify with MusicID-File, you must create *file information object*, called a FileInfo. Each FileInfo object stores the Track metadata to be used for identification. When the identification query completes, MusicID-File stores the query results in the FileInfo object.

When creating a FileInfo object, you can specify unique string identifier for the media file the FileInfo represents. Once created, you can then use `gnsdk_musicidfile_query_fileinfo_get_by_ident` to retrieve the FileInfo object from a Track Response GDO. See [Getting Results from TrackID and AlbumID](#)

During an identification query, MusicID-File uses the FileInfo metadata (and optionally audio fingerprint data ) to match each Track to an Album. When finished, it returns a Response GDO of type `GNSDK_GDO_TYPE_RESPONSE_ALBUM`. This GDO contains one or more Albums of type `GNSDK_GDO_TYPE_ALBUM`. Each Album contains one or more matching Tracks of type `GNSDK_GDO_TYPE_TRACK`.

FileInfo objects are freed when you release the MusicID-File query handle.

```

/*
 * Create a FileInfo object and add to query
 */

```

```
gnsdk_musicidfile_query_fileinfo_create(  
    query_handle, /* Query handle to create the FileInfo object */  
    "data/01_stone_roses.wav", /* Unique string identifier for media  
file */  
    GNSDK_NULL, /* No callback function for status and progress */  
    GNSDK_NULL, /* No callback data */  
    &fileinfo_handle /* Pointer to receive the created FileInfo object  
*/  
);
```

### 3.11.3.6 Setting MusicID-File Query Options

To set an option for your MusicID-File query, use `gnsdk_musicidfile_query_option_set`. For example, the following sets the option to use 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.

Besides local lookup, there are options for returning the following: alternate names for contributor data, classical music data, fetching content (for example, images) data, external IDs, playlist data, and mood and tempo data. You can also set the preferred language and thread priority. For LibraryID, you can also set the batch size. See the API reference for a complete list and more information.

```
gnsdk_musicidfile_query_option_set(  
    query_handle,  
    GNSDK_MUSICIDFILE_OPTION_LOOKUP_MODE,  
    GNSDK_LOOKUP_MODE_LOCAL  
);
```

### 3.11.3.7 Setting Media File Identification Process

To identify a media files, you must specify the kind of identification process you want MusicID-File to use. There are two main processes:

- Fingerprinting
- File Metadata

For TrackID and AlbumID, you set the identification process before making the query. For LibraryID, you specify the process using callbacks after the query finishes. .

## Fingerprinting

The MusicID-File fingerprinting APIs use audio data as an identification mechanism. This enables MusicID-File to perform identification based on the audio itself, instead of using only the media file metadata.

For TrackID and Album ID, use the fingerprinting APIs before running the identification query. For LibraryID, use the fingerprinting APIs during the callback `gnsdk_musicidfile_callback_get_fingerprint_fn`.

Fingerprinting APIs are:

**`gnsdk_musicidfile_fileinfo_fingerprint_begin`**: Initializes fingerprint generation for a FileInfo handle.

**`gnsdk_musicidfile_fileinfo_fingerprint_end`**: Finalizes fingerprint generation for a FileInfo handle.

**`gnsdk_musicidfile_fileinfo_fingerprint_write`**: Provides uncompressed audio data to a FileInfo handle for fingerprint generation.

Example:

```
gnsdk_musicidfile_fileinfo_fingerprint_begin(  
    fileinfo_handle, /* FileInfo handle for fingerprint */  
    11025, /* Sample frequency: 11 kHz, 22 kHz, or 44 kHz */  
    16, /* Sample rate : 8-bit, 16-bit, or 32-bit bytes per sample */  
    1 /* Number of channels (1 or 2) */  
);  
  
gnsdk_musicidfile_fileinfo_fingerprint_write(  
    fileinfo_handle, /* FileInfo handle for the fingerprint */  
    pcm_audio, /* Pointer to audio data buffer that matches audio  
format  
                set in gnsdk_musicidfile_FileInfo_fingerprint_begin */  
    num_bytes, /* Size of audio data buffer (in bytes) */  
    &complete /* Pointer to receive whether the fingerprint  
generation has received enough audio data */  
);
```



## Setting and Getting FileInfo Metadata

Use `gnsdk_musicidfile_fileinfo_metadata_set` and `gnsdk_musicidfile_fileinfo_metadata_get` to set and get metadata information for a FileInfo object. MusicID-File will not process FileInfo objects that do not contain metadata. See the API reference for a complete list of metadata you can set and get.

### Example:

```
/* Set Artist */
gnsdk_musicidfile_fileinfo_metadata_set(
    fileinfo_handle,
    GNSDK_MUSICIDFILE_FILEINFO_VALUE_ALBUMARTIST,
    "Kardinal Offishall"
);

/* Set Album Title */
gnsdk_musicidfile_fileinfo_metadata_set(
    fileinfo_handle,
    GNSDK_MUSICIDFILE_FILEINFO_VALUE_ALBUMTITLE,
    "Quest for Fire"
);
```

### 3.11.3.8 Performing the Identification Query

Each identification process has its own query function:

**TrackID:** `gnsdk_musicidfile_query_do_trackid`

**AlbumID:** `gnsdk_musicidfile_query_do_albumid`

**LibraryID:** `gnsdk_musicidfile_query_do_libraryid`

Each API takes a MusicID-File query handle and a set of query flags. These flags indicate options specific to that query. These include:

**Asynchronous:** Processes MusicID-File on a separate thread and returns immediately

**Default options:** Use default MusicID-File processing options - request a single, best album match. See the GNSDK\_MUSICIDFILE\_QUERY\_FLAG\_DEFAULT define in the API reference for more information

**No threads:** Disallows MusicID-File from creating threads for background gathering of fingerprints and metadata

**Album responses:** Only album matches are returned (default)

**Album or contributor match responses:** Album and contributor matches are returned

**Return all:** Have MusicID-File return all results found for each given FileInfo

**Return single:** Have MusicID-File return the single best result for each given FileInfo (default)

See the API reference for more information on the defines for these options.

You can get a query's status at any time with `gnsdk_musicidfile_query_status`.

Example:

```
/* Set options and perform the Query */
gnsdk_uint32_t midf_options =
    GNSDK_MUSICIDFILE_QUERY_FLAG_RETURN_ALL |
    MIDF_QUERY_FLAG |
    GNSDK_MUSICIDFILE_QUERY_FLAG_NO_THREADS;

gnsdk_musicidfile_query_do_trackid(query_handle, midf_options);
```

### 3.11.3.9 Getting Results with TrackID and AlbumID

After making your query, you can retrieve the Track child GDO, and then use `gnsdk_manager_gdo_value_get` with `GNSDK_MUSICIDFILE_GDO_VALUE_IDENT` to retrieve the identifier value for the matching FileInfo.

You can use the following call to make sure processing has completed within a certain time period

```
gnsdk_musicidfile_query_wait_for_complete(
    query_handle,
```

```
GNSDK_MUSICIDFILE_TIMEOUT_INFINITE, /* Time to wait in msec or
until done (INFINITE) */
GNSDK_NULL /* Pointer to error returned on completion */
);
```

Next, use the following call to get the number of FileInfo objects:

```
gnsdk_uint32_t count = 0;
gnsdk_musicidfile_query_fileinfo_count(
query_handle,
&count);
```

Once you have the count, and it is greater than 0, you can use that as an index to retrieve FileInfo objects. If more than 1, typically this would be done in a loop:

```
for (i = 0; i < count; i++)
{
    gnsdk_musicidfile_query_fileinfo_get_by_index(
        query_handle,
        i,
        &fileinfo_handle);
    ...
}
```

### Example:

```
/* Find the number of matched tracks in an album GDO */
error = gnsdk_manager_gdo_child_count(
    album_gdo, /* Album GDO from the Response GDO */
    GNSDK_GDO_CHILD_TRACK_MATCHED, /* Key to get the album */
    &count); /* Pointer to receive the result */
/* Iterate over the matched tracks */
for (i=1; i<=count; i++)
{
    /* Get the matched track */
    error = gnsdk_manager_gdo_child_get(
        album_gdo, /* Album Response GDO handle*/
        GNSDK_GDO_CHILD_TRACK_MATCHED, /* Key to get the matched track GDO */
        i, /* Instance, starts with 1 */
        &track_gdo); /* Track GDO returned */

    /* Get the identifier for this matched track */
    error = gnsdk_manager_gdo_value_get(
        track_gdo, /* Track Response GDO handle */
        GNSDK_MUSICIDFILE_GDO_VALUE_IDENT, /* Key to get the full GDO */
        1, /* Retrieve n'th instance of the value (1-based) */
```

```

    &ident_str); /* Pointer to receive the identifier string */

    /* Get the FileInfo using media file identifier */
    error = gnsdk_muscidfile_query_fileinfo_get_by_ident(
        track_gdo, /* Track Response GDO handle */
        ident_str, /* String identifier of FileInfo to retrieve */
        &fileinfo_handle); /* Pointer to receive FileInfo handle */

    /* Additional processing to retrieve track metadata from FileInfo ) */
    ...
}

```

Once you have the FileInfo object, you should check its status to make sure it completed successfully. If it did, you can then get its Response GDO:

```

gnsdk_muscidfile_fileinfo_status_t fileinfo_status = gnsdk_
muscidfile_fileinfo_unprocessed;
gnsdk_error_info_t* p_error_info = GNSDK_NULL;
gnsdk_gdo_handle_t results_gdo = GNSDK_NULL;

/* Check status. If ok, get Response GDO */
gnsdk_muscidfile_fileinfo_status(fileinfo, &fileinfo_status, &p_
error_info);
if (gnsdk_muscidfile_fileinfo_error != fileinfo_status)
{
    if ((gnsdk_muscidfile_fileinfo_result_single == fileinfo_status) ||
        (gnsdk_muscidfile_fileinfo_result_all == fileinfo_status))
    {
        gnsdk_muscidfile_fileinfo_get_response_gdo(fileinfo, &response_
gdo);
        ... /* Parse Response GDO */
    }
}
}

```

### 3.11.3.10 Getting Results with LibraryID

With LibraryID, results are returned in a results-available callback. Note that the Response GDO is passed to the callback.

```

static gnsdk_void_t GNSDK_CALLBACK_API
_result_available_callback(
    const gnsdk_void_t* user_data,
    gnsdk_muscidfile_query_handle_t query_handle,
    gnsdk_gdo_handle_t response_gdo,
    gnsdk_uint32_t current_album,

```

```
gnsdk_uint32_t  total_albums,  
gnsdk_bool_t*   p_abort  
)  
{  
    ...Parse Response GDO  
} /* _result_available_callback */
```

### 3.11.3.11 Releasing Resources and Shutting Down

In addition to the usual releasing of resources (user handle, locale handle, GDOs, and so on.), the following are specific to MusicID-File. You should **not** call shutdown if the initialize function call fails. Releasing the MusicID-File query handle also releases all associated FileInfo objects. For each module you initialize, you can either call the shutdown for that module or just call GNSDK Manager shutdown - `gnsdk_manager_shutdown` - which will shut down all libraries.

```
/* Release MusicID-File - releases associated FileInfo objects */  
gnsdk_musicidfile_query_release(query_handle);  
gnsdk_musicidfile_shutdown();
```

### 3.11.3.12 Advanced Music Identification using MusicID-File

MusicID-File provides three processing methods enabling advanced file-based media recognition. Each method utilizes the same population and result management APIs, so which method to use is determined by the application's requirement at the time of processing. MusicID-File returns a GDO for each result - providing as many as needed.

#### TrackID

The `gnsdk_musicidfile_query_do_trackid()` API provides TrackID processing.

#### ***Example: Implementing TrackID***

Sample Application: `musicid_file_trackid/main.c`

Application Steps:

1. Authenticate caller, initialize GNSDK Manager, enable logging, and initialize local lookup.
2. Initialize MusicID File and DSP, get a User handle and load a locale.
3. Create a MusicID-File query handle
4. Get a FileInfo object for each media file and add to query handle.
5. Fingerprint and set metadata for each media file in its FileInfo object.
6. Perform two MusicID-File TrackID lookups, first with RETURN\_SINGLE, second with RETURN\_ALL.
7. Display results.
8. Release resources and shutdown GNSDK and MusicID File module.

**Sample output:**

```
GNSDK Product Version   : 3.05.0.721    (built 2013-04-02 22:29-0700)
-----TrackID with 'RETURN_SINGLE' option:-----
Warning: input file does not contain enough data to generate a
fingerprint:
data\kardinal_offishall_01_3s.wav

Printing results for 6 files:

*File 1 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Crimson Tonight [Live]

*File 2 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Crimson Tonight [Live]

*File 3 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Crimson Tonight [Live]

*File 4 of 6*
```

```
Single result.
Album count: 1
Match 1 - Album title:    Sugar Baby

*File 5 of 6*

Single result.
Album count: 1
Match 1 - Album title:    Quest For Fire

*File 6 of 6*

Single result.
Album count: 1
Match 1 - Album title:    Firestarter Volume 1 - Quest For Fire

-----TrackID with 'RETURN_ALL' option:-----
Warning: input file does not contain enough data to generate a
fingerprint:
data\kardinal_offishall_01_3s.wav

Printing results for 6 files:

*File 1 of 6*

Multiple results.
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]

*File 2 of 6*

Multiple results.
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]

*File 3 of 6*

Multiple results.
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]

*File 4 of 6*

Multiple results.
```

```

Album count: 3
Match 1 - Album title:      Country Blues
Match 2 - Album title:      SÃ¥nger Om KÃ¶rlek [Disc 2]
Match 3 - Album title:      His Folkways Years 1963-1968 [Disc 2]

*File 5 of 6*

Multiple results.
Album count: 1
Match 1 - Album title:      Quest For Fire

*File 6 of 6*

Multiple results.
Album count: 6
Match 1 - Album title:      Firestarter Volume 1 - Quest For Fire
Match 2 - Album title:      New English File Advanced
Match 3 - Album title:      Jazzblues
Match 4 - Album title:      èµ·ã  ã |ã <ã,%ã ¯ ã,<ã ¾ã §è<±ã¼šè©±ã ¾ã
, <ã ¨ã ¨ç ¸ç:'ã,³2
Match 5 - Album title:      ç¬¬66ã>ž ä,ã>½èªžææã@šè©|é"" 3ç'š
Match 6 - Album title:      Quest For Fire

```

## AlbumID

The `gnsdk_musicidfile_query_do_albumid()` API provides AlbumID processing.

### *Example: Implementing AlbumID*

Sample Application: `musicid_file_albumid/main.c`

Application steps:

1. Authenticate caller, initialize GNSDK Manager, enable logging, and initialize local lookup
2. Initialize MusicID File and DSP, get a User handle and load a locale
3. Create a MusicID-File query handle
4. Get a FileInfo object for each media file and add to query handle
5. Fingerprint and set metadata for each media file in its FileInfo object



6. Perform two MusicID-File Album ID lookups, first with RETURN\_SINGLE, second with RETURN\_ALL
7. Display results
8. Release resources and shutdown GNSDK and MusicID File module

**Sample output:**

```
GNSDK Product Version : 3.05.0.721 (built 2013-04-02 22:29-0700)
-----AlbumID with 'RETURN_SINGLE' option:-----
Warning: input file does contain enough data to generate a
fingerprint:
data\kardinal_offishall_01_3s.wav

Printing results for 6 files:

*File 1 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Quest For Fire

*File 2 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Quest For Fire

*File 3 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Sugar Baby

*File 4 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Crimson Tonight [Live]

*File 5 of 6*

  Single result.
  Album count: 1
  Match 1 - Album title:    Crimson Tonight [Live]
```

\*File 6 of 6\*

Single result.

Album count: 1

Match 1 - Album title: Crimson Tonight [Live]

-----AlbumID with 'RETURN\_ALL' option:-----

Warning: input file does contain enough data to generate a fingerprint:

data\kardinal\_offishall\_01\_3s.wav

Printing results for 6 files:

\*File 1 of 6\*

Multiple results.

Album count: 6

Match 1 - Album title: Firestarter Volume 1 - Quest For Fire

Match 2 - Album title: New English File Advanced

Match 3 - Album title: Jazzblues

Match 4 - Album title: èµ·ă ä |ă <ă,%ă<sup>~</sup> ä,<ă ¼ă Şè<±ă¼šè©±ă ¼ă

,<ă "ă "ç·'çç'â,³2

Match 5 - Album title: ç¬¬66â>ž ä,â>½èªžæœâ@šè©|é"" 3ç'š

Match 6 - Album title: Quest For Fire

\*File 2 of 6\*

Multiple results.

Album count: 2

Match 1 - Album title: Firestarter Volume 1 - Quest For Fire

Match 2 - Album title: Quest For Fire

\*File 3 of 6\*

Multiple results.

Album count: 3

Match 1 - Album title: Country Blues

Match 2 - Album title: SĂŸnger Om KĂrlek [Disc 2]

Match 3 - Album title: His Folkways Years 1963-1968 [Disc 2]

\*File 4 of 6\*

Multiple results.

```
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]

*File 5 of 6*

Multiple results.
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]

*File 6 of 6*

Multiple results.
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]
```

## LibraryID

The `gnsdk_musicidfile_query_do_libraryid()` API provides LibraryID processing.

Response GDOs from LibraryID queries are only available via status and result callbacks. This is because all LibraryID GDOs are freed after the process finishes.

### *Example: Implementing LibraryID*

Sample Application: `musicid_file_libraryid/main.c`

Application steps:

1. Authenticate caller, initialize the GNSDK, enable logging, and initialize local lookup
2. Initialize DSP and MusicID File, load a locale, and get a User handle
3. Create a MusicID-File query handle
4. Perform LibraryID query, passing callbacks
5. Fingerprint media file in callback
6. Add metadata to media file in callback
7. Handle and display statuses and results in callbacks
8. Release resources and shutdown GNSDK and MusicID-File

**Sample output:**

```
GNSDK Product Version   : 3.05.0.798    (built 2013-05-08 16:09-0700)

MID-File Status: 1 of 6 - fileinfo_processing_begin - data\01_stone_
roses.wav

MID-File Status: 2 of 6 - fileinfo_processing_begin - data\04_stone_
roses.wav

MID-File Status: 3 of 6 - fileinfo_processing_begin - data\Dock Boggs
- Sugar Baby - 01.wav

MID-File Status: 4 of 6 - fileinfo_processing_begin - data\Kardinal
Offishall - Quest For Fire - 15 - Go Ahead Den.wav

MID-File Status: 5 of 6 - fileinfo_processing_begin - data\kardinal_
offishall_01_3s.wav

MID-File Status: 6 of 6 - fileinfo_processing_begin - data\stone roses
live.wav

MID-File Status: 1 of 6 - fileinfo_query - data\Kardinal Offishall -
Quest For Fire - 15 - Go Ahead Den.wav

MID-File Status: 1 of 6 - fileinfo_query - data\Kardinal Offishall -
Quest For Fire - 15 - Go Ahead Den.wav

MID-File Status: 2 of 6 - fileinfo_query - data\Dock Boggs - Sugar
Baby - 01.wav

MID-File Status: 2 of 6 - fileinfo_query - data\Dock Boggs - Sugar
Baby - 01.wav

MID-File Status: 3 of 6 - fileinfo_query - data\04_stone_roses.wav

MID-File Status: 4 of 6 - fileinfo_query - data\01_stone_roses.wav
Warning: input file does not contain enough data to generate a
fingerprint:
data\kardinal_offishall_01_3s.wav

MID-File Status: 5 of 6 - fileinfo_query - data\stone roses live.wav

MID-File Status: 2 of 6 - fileinfo_query - data\Dock Boggs - Sugar
Baby - 01.wav

MID-File Result:
```

```
Album count: 1
Match 1 - Album title:    Crimson Tonight [Live]

MID-File Result:
Album count: 1
Match 1 - Album title:    Sugar Baby

MID-File Result:
Album count: 1
Match 1 - Album title:    Firestarter Volume 1 - Quest For Fire

MID-File Status: 1 of 6 - (null) - data\Kardinal Offishall - Quest For
Fire - 15 - Go Ahead Den.wav

MID-File Status: 2 of 6 - (null) - data\Dock Boggs - Sugar Baby -
01.wav

MID-File Status: 3 of 6 - (null) - data\04_stone_roses.wav

MID-File Status: 4 of 6 - (null) - data\01_stone_roses.wav

MID-File Status: 5 of 6 - (null) - data\stone roses live.wav

MID-File Status: 6 of 6 - (null) - data\kardinal_offishall_01_3s.wav
```

### **3.11.4** *Identifying Streaming Music*

The functionality for identifying streaming music is contained in the library `gnsdk_muscid_stream`.

In general, identification requires up to seven seconds of audio. If there is not enough audio available to make the identification, the system waits until enough audio is received. The identification process triggers callbacks and completes asynchronously.

You can cancel the identification process outside of a callback using `gnsdk_muscidstream_channel_identify_cancel()`. Canceling calls the identify error delegate method with an "aborted" error. Canceling does not stop audio processing. Your application can restart the identification process after canceling.

When canceling a user-initiated query, a result may have already been returned internally. Your application can decide whether or not to show this result to the user.

#### 3.11.4.1 MusicID Stream Callbacks

MusicID Stream requires callbacks to receive identification results. When MusicID Stream receives enough audio, it will perform the identification and send the result back to the appropriate callback. The callbacks are set when creating the channel:

```
/* Declare variables for callbacks and resulting channel */
gnsdk_musicidstream_channel_handle_t channel_handle = GNSDK_NULL;
gnsdk_musicidstream_callbacks_t      callbacks      = {0};
gnsdk_error_t                        error          = GNSDK_SUCCESS;
int                                  rc              = 0;

/* Set necessary callbacks for the application. */
callbacks.callback_status              = GNSDK_NULL;
callbacks.callback_processing_status   = GNSDK_NULL;
callbacks.callback_identifying_status =
    _musicidstream_identifying_status_callback;
callbacks.callback_result_available    =
    _musicidstream_result_available_callback;
callbacks.callback_error               =
    _musicidstream_completed_with_error_callback;

/* Create the channel handle */
error = gnsdk_musicidstream_channel_create(
    user_handle,
    gnsdk_musicidstream_preset_radio,
    &callbacks,
    GNSDK_NULL,
    &channel_handle
);
```

Once you have created your channel handle, you read in the data from your source, and write it out.

```
/* Give audio data information for the channel */
error = gnsdk_musicidstream_channel_audio_begin(
    channel_handle, 44100, 16, 2 );
```

```
/* Launch the identification request */
error = gnsdk_musicidstream_channel_identify(channel_handle);

/* Read in data and process */
...
/* write audio to the fingerprinter */
error = gnsdk_musicidstream_channel_audio_write(
    channel_handle, pcm_audio, read_size);
...
/* When we are done, end the stream */
error = gnsdk_musicidstream_channel_audio_end(channel_handle);

/* Wait for the identification to finish so we actually get results */
gnsdk_musicidstream_channel_wait_for_identify(channel_handle, GNSDK_
MUSICIDSTREAM_TIMEOUT_INFINITE);

/* Close the channel. */
gnsdk_musicidstream_channel_release(channel_handle);
```

### 3.11.4.2 Example: Identifying Streaming Music

Sample Application: musicid\_stream\_manual/main.c

Application Steps:

1. Authenticate User and initialize Manager, SQLite, LocalStream, DSP, MusicID, and other modules as necessary.
2. Create a Music-ID Stream channel and write audio.
3. Perform identification and display results.
4. Release resources and shutdown GNSDK and modules.

This application reads data from a .wav file and includes examples of necessary callbacks.

**Sample output:**

```
GNSDK Product Version : v3.05.0.721 (built 2013-04-02 22:29-0700)

*****Sample MID-Stream Query*****
Match count: 2
```

```
Title: A Ghost is Born
Title: Mensajes
Final album:
Title: A Ghost is Born
```

### 3.11.5 Implementing Audio Suitability Processing (ASP)

Implementing audio suitability processing, that is, determining if the audio you are capturing is of high enough quality to attempt an identification, can be done with both `MusicID` (prior to queries) and `MusicIdStream` (prior to listening).

#### 3.11.5.1 Setting ASP for Queries

To turn on ASP, set a GNSDK music query option to `true`:

```
/* enable audio suitability processing */
error = gnsdk_musicid_query_option_set(query_handle, GNSDK_MUSICID_
OPTION_ENABLE_AUDIO_SUITABILITY_PROCESSING, GNSDK_VALUE_TRUE);
```

Before you make a query, you can check for suitability:

```
/* check if audio data is suitable for query */
error = gnsdk_musicid_query_fingerprint_info_get(query_handle, GNSDK_
MUSICID_AUDIO_INFO_SUITABLE_FOR_QUERY, &value);
```

#### 3.11.5.2 Setting ASP for MusicID Stream

To turn on ASP for stream, set a GNSDK music stream channel option to `true`:

```
/* enable audio suitability processing */
error = gnsdk_musicidstream_channel_option_set(channel_handle, GNSDK_
MUSICIDSTREAM_OPTION_ENABLE_AUDIO_SUITABILITY_PROCESSING, GNSDK_VALUE_
TRUE);
```

#### Note About Preset Selection

When you create a channel (`gnsdk_musicidstream_channel_create`) you have to pass a channel *preset*, either for radio or microphone. Radio is designed for medium to good quality audio with no background noise, such as



people talking. Microphone is designed for low quality audio received via a microphone that may have high levels of background noise such as people talking. If you select the radio preset and enable ASP, you will need to load a file to aid in suitability processing:

```
/* Use Radio classifier for suitability processing */
error = gnsdk_dsp_storage_location_set(GNSDK_DSP_CLASSIFIER_MODEL_
STORAGE_LOCATION, "location/of/gn_dsp/gcl/file");
```

When you begin streaming, If audio is not suitable, an error message is returned:

```
/* MusicID Stream Error Callback */
gnsdk_void_t GNSDK_CALLBACK_API
_musidstream_completed_with_error_callback(gnsdk_void_t* callback_
data, gnsdk_musidstream_channel_handle_t channel_handle, const
gnsdk_error_info_t* p_error_info)
{
    if (p_error_info->error_code == MIDSWARN_UnsuitableAudio)
    {
        /* Audio not suitable for music identification */
    }
}
```



**Note:** There are other options available that can give you finer control over defining, controlling, and assessing suitability. To implement this, you should talk to your Gracenote customer representative.

### **3.11.6** *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 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 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.

#### 3.11.6.1 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.

#### 3.11.6.2 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.

### 3.11.6.3 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.

### 3.11.6.4 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.

### 3.11.6.5 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.

### 3.11.6.6 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.

### 3.11.6.7 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.

## 3.11.7 *Identifying Music Using Batch Processing*

Batch processing allows multiple queries to be performed in a single request to Gracenote Service. A single batch request can contain different query types, and

each query can have different inputs. For example, a batch lookup can have a query for finding matches with text inputs in addition to finding matches based on fingerprint input.

#### 3.11.7.1 Creating a Batch Query

You create queries the same way as any other MusicID query, but instead of invoking a find function on a single query, you add the query to a batch lookup handle, and then call a batch execute function. Each query requires a unique string identifier. The identifier allows the application to associate the query with its results. Creating a batch query with an identifier already in use results in an error. Gracenote recommends batches between 20 and 50 queries. In creating your batch queries please note the following:

- Batches containing 20 queries offer a measurable reduction in average query processing time as well as uplink and downlink transmission size without much of an increase to peak memory usage.
- Batches containing approximately 50 queries offers some reduction in average query processing time as well as uplink and downlink data transmissions, but the peak memory usage is higher.
- Batches greater than 50 queries often increases average query processing time as well as uplink and downlink data transmissions and require larger peak memory usage.
- Batch sizes of one are discouraged. It is more efficient to use a standard query if only one result is required.

The following example creates an array of 5 text queries, and then calls the batch function `gnsdk_musicid_batch_query_create()` to create the query, and `gnsdk_musicid_batch_query_set_text()` to set query text:

```
/*
 NOTE: Return code checking has been removed for brevity.  Always
 check GNSDK return codes.
 Create the batch query handle.
*/
gnsdk_musicid_batch_create(
    user_handle,      /* From program initialization */
    GNSDK_NULL,      /* User callback function */
```

```
    GNSDK_NULL,          /* Optional data to be passed to the callback */
    &batch_handle        /* Batch handle returned */
);

/*
   Add queries to the batch query handle. This section of code adds
   queries based on the batch handle,
   and within that handle, on the unique name of the individual query.
*/
/* create individual queries for the batch handle, each query must
   have a unique identifier */
/* Track 1 */
gnsdk_muscid_batch_query_create(batch_handle, "track_1");
gnsdk_muscid_batch_query_set_text(batch_handle, "track_1", GNSDK_
MUSCID_FIELD_TRACK_ARTIST, "Jesse Cook");
gnsdk_muscid_batch_query_set_text(batch_handle, "track_1", GNSDK_
MUSCID_FIELD_ALBUM, "Ultimate Jesse Cook (Disc 2)");
gnsdk_muscid_batch_query_set_text(batch_handle, "track_1", GNSDK_
MUSCID_FIELD_TITLE, "On Walks The Night");

/* Track 2 */
gnsdk_muscid_batch_query_create(batch_handle, "track_2");
gnsdk_muscid_batch_query_set_text(batch_handle, "track_2", GNSDK_
MUSCID_FIELD_TRACK_ARTIST, "The Cure");
gnsdk_muscid_batch_query_set_text(batch_handle, "track_2", GNSDK_
MUSCID_FIELD_ALBUM, "Join The Dots: B-Sides and Rarities 1978-2001
(Box Set CD 1)");
gnsdk_muscid_batch_query_set_text(batch_handle, "track_2", GNSDK_
MUSCID_FIELD_TITLE, "A Man Inside My Mouth");

/* additional queries can be created based on other inputs such as CD
   TOC, fingerprint and identifier.*/
```

### 3.11.7.2 Performing a Batch Query and Retrieving Data

After the batch query is set, you retrieve the results from all queries with one call to `gnsdk_muscid_batch_find_matches()` or `gnsdk_muscid_batch_find_albums()`. The individual queries are executed in one request to Gracenote Service. You retrieve the responses from the batch handle using `gnsdk_muscid_batch_response_get()`, based on the unique identifier

for the query. Queries executed in batch are limited to a single response. Because of this, `GNSDK_MUSICID_OPTION_RESULT_SINGLE` is automatically set and cannot be unset. The following code executes the batch query based on the queries in the previous snippet, and retrieves the individual response GDOs: It then releases the batch query handle:

```
/* Perform batch query using the batch query handle. */
gnsdk_musicid_batch_find_matches(batch_handle);

/*
   At this point all responses have been created with one call to the
   database.  You can now retrieve response GDO data
   based on the unique query id.
*/

/* Track 1 */
gnsdk_musicid_batch_response_get(batch_handle, "track_1", &response_
gdo);
/* process Track 1 response */

/* Track 2 */
gnsdk_musicid_batch_response_get(batch_handle, "track_2", &response_
gdo);
/* process Track 1 response */

/*
   After you have completed processing, release any query handles or
   GDOs as required.
   Following completion of batch queries, you must also release the
   batch query handle.
*/
gnsdk_musicid_batch_release(batch_handle);
```

### 3.11.7.3 Setting Batch Lookup Options

Options set on a batch lookup affect all queries added to the batch lookup handle. Options are the same for batch queries as for non-batched queries. The following code sets the lookup mode for the batch queries to online nocache mode.

```
/* Set option for online lookup */
gnsdk_musicid_batch_option_set(batch_handle, GNSDK_MUSICID_OPTION_
LOOKUP_MODE, GNSDK_LOOKUP_MODE_ONLINE_NOCACHE);
```

### 3.11.7.4 Examining Batch Functions

The following table contains the listing of batch query functions available. Consult `gnsdk_musicid.h` for further information.

Function Name	Description
<code>gnsdk_musicid_batch_clear()</code>	Removes all individual queries from a MusicID batch query.
<code>gnsdk_musicid_batch_create()</code>	Creates a MusicID batch query handle.
<code>gnsdk_musicid_batch_find_albums()</code>	Performs a MusicID batch query for album results.
<code>gnsdk_musicid_batch_find_matches()</code>	Performs a MusicID batch query for matches of the best fit type (albums or contributors).
<code>gnsdk_musicid_batch_option_get()</code>	Retrieves an option for a given MusicID batch query handle.
<code>gnsdk_musicid_batch_option_set()</code>	Sets an option for a given MusicID batch query handle. Options set on the batch handle apply to individual queries processed in the batch.
<code>gnsdk_musicid_batch_query_add_toc_offset()</code>	Sets CD TOC offset values for an individual query.
<code>gnsdk_musicid_batch_query_create()</code>	Creates a new individual query to be processed as part of the MusicID batch query.
<code>gnsdk_musicid_batch_query_set_fp_data()</code>	Sets externally-generated fingerprint data for an individual query.

Function Name	Description
<code>gnsdk_musicid_batch_query_set_gdo()</code>	Sets GDO input for an individual query.
<code>gnsdk_musicid_batch_query_set_text()</code>	Sets text inputs for an individual query.
<code>gnsdk_musicid_batch_query_set_toc_string()</code>	Sets CD TOC input for an individual query.
<code>gnsdk_musicid_batch_release()</code>	Invalidates and releases resources for a given MusicID batch query handle.
<code>gnsdk_musicid_batch_response_get()</code>	Gets an individual query's response from a batch query using it's unique identifier.
<code>gnsdk_musicid_batch_set_locale()</code>	Sets a locale to be used as input for all queries added to a MusicID batch query.

### 3.11.7.5 Example: Executing Batch Lookups

Sample Application: `musicid_lookup_batch_matches_text/main.c`

Application Steps:

1. Authenticate User and initialize Manager and other modules as necessary.
2. Create two batch query arrays.
3. Perform the batch queries and trace through response GDOs for results.
4. Release resources and shutdown GNSDK and modules.

### 3.11.8 Collaborative Artists Best Practices

The following topic provides best practices for handling collaborations in your application.



### 3.11.8.1 Handling Collaborations when Processing a Collection

When looking up a track using a text-based lookup, such as when initially processing a user's collection, use the following best practices:

- If the input string matches a single artist in the database, such as "Santana," associate the track in the application database with the single artist.
- If the input string matches a collaboration in the database, such as "Santana featuring Rob Thomas," associate the track in the application database with the primary collaborator and the collaboration. In this case, the Contributor, "Santana featuring Rob Thomas," will have a Contributor child, "Santana," and the track should be associated with "Santana" and "Santana featuring Rob Thomas."
- If the input string is a collaboration, but does not match a collaboration in the database, GNSDK attempts to match on the primary collaborator in the input, which would be "Santana" in this example. If the primary collaborator matches an artist in the database, the result will be the single artist. There will be an indication in the result that only part of the collaboration was matched. Associate the track in the application database with the single artist and with the original input string.

### 3.11.8.2 Displaying Collaborations during Playback

When determining what should be displayed during playback of music, use the following best practices:

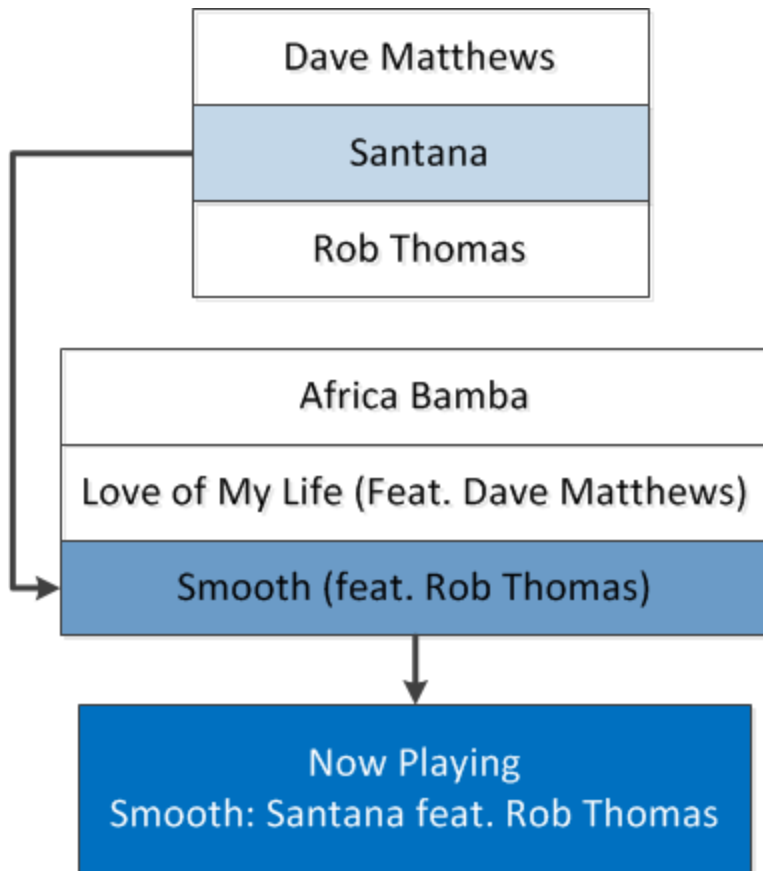
- When a track by a single artist is playing, your application should display the Gracenote normalized text string. For example, when a track by Santana is playing, "Santana" should be displayed.
- When a track by a collaboration is playing, and GNSDK has matched the collaboration, the application should display the collaboration name. For example, when a track by "Santana featuring Rob Thomas" is playing, the collaboration name "Santana featuring Rob Thomas" should be displayed.
- When a track by a collaboration is playing, but only part of the collaboration was matched, Gracenote recommends that you display the original tag data

for that track during playback. For example, when a track by “Santana featuring Unknown Artist” is playing, but only “Santana” was matched, the collaboration name “Santana featuring Unknown Artist” should be displayed. Gracenote recommends that you do not overwrite the original tag data.

### 3.11.8.3 Displaying Collaborations in Navigation

When creating navigation in your application, use the following best practices:

- If the user is navigating through the interface, and comes to “Santana” in a drop-down list, all tracks by “Santana” should be displayed, including tracks on which Santana is the primary collaborator. The list should be created using the associations that you created during the initial text-lookup phase. If the user selects “Play songs by Santana,” all songs by Santana and songs on which Santana is the primary collaborator can be played.
- Gracenote does not recommend that collaborations appear in drop-down lists of artists. For example, don’t list “Santana” and “Santana featuring Rob Thomas” in the same drop-down list. Instead, include “Santana” in the drop-down list.



#### 3.11.8.4 Handling Collaborations in Playlists

When creating a playlist, if the user is able to select a collaboration as a seed, then only songs by that collaboration should be played. For example, if the user selects “Santana featuring Rob Thomas” as a seed for a playlist, they should only hear songs by that specific collaboration. This only applies to playlists of the form “Play songs by <artist>.” It does not apply to “More Like This” playlists, such as “Play songs like <artist>,” which use Gracenote descriptors to find similar artists.

#### 3.11.9 Navigating Music GDOs

Top-level music GDOs generally represent Albums and Tracks. An album or track query can return a response GDO containing 0-n matches (child GDOs). For

example, a track query could return multiple album matches since the track may exist on more than one album. In this case, to display metadata information for one album, the end-user or your application needs to select the child GDO representing a specific album. For more information on managing multiple results, see *"About Gracenote Data Objects (GDOs)" on page 138*.

Queries can return matches (child GDOs) containing either full or partial metadata results. For information on handling this, see *"Full and Partial Metadata Results" on page 140*.

The following code assumes that an initial query has already been performed and has returned a response GDO. A GDO's child objects are enumerated with an ordinal index, starting from 1 (not 0) for the first child object. From the response GDO, the following code accesses the first child album GDO.

```
...Perform album query and get response GDO

gnsdk_uint32_t  ordinal_index = 1;

/*
 * Get first child album GDO from response GDO
 */
gnsdk_gdo_handle_t  album_gdo = GNSDK_NULL;
gnsdk_manager_gdo_child_get(response_gdo, GNSDK_GDO_CHILD_ALBUM,
ordinal_index, &album_gdo);

/*
 * Get first child track GDO from album GDO
 */
gnsdk_gdo_handle_t  track_gdo = GNSDK_NULL;
gnsdk_manager_gdo_child_get(album_gdo, GNSDK_GDO_CHILD_TRACK, ordinal_
index, &track_gdo);
```

Note that the **ordinal index** of a track GDO within an album GDO has no relationship to its track number within the album itself. To get the actual track number, use the GNSDK\_GDO\_VALUE\_TRACK\_NUMBER value key:

```
gnsdk_cstr_t  value = GNSDK_NULL;
gnsdk_manager_gdo_value_get(track_gdo, GNSDK_GDO_VALUE_TRACK_NUMBER,
1, &value);
```

Sample Application: musicid\_gdo\_navigation/main.c

This application uses MusicID to look up Album GDO content, including Album artist, credits, title, year, and genre. It demonstrates how to navigate the album GDO that returns track information.

### 3.11.9.1 Accessing Classical Music Metadata

To retrieve classical music metadata, set a query handle with the applicable query option set to True to access the metadata and *automatically render it to the XML output*. For an overview of classical music metadata, see "[Classical Music Metadata](#)" on page 6.

The following table lists the applicable query functions and options for the GNSDK music modules.

Product	Query Function	Query Option
MusicID	gnsdk_musicid_query_option_set gnsdk_musicid_query_option_get	GNSDK_MUSICID_OPTION_ENABLE_CLASSICAL_DATA
MusicID-File	gnsdk_musicidfile_query_option_get	GNSDK_MUSICIDFILE_OPTION_ENABLE_CLASSICAL_DATA



**Note:** Your application must be licensed to access this metadata. Contact Gracenote for more information.

### Classical Album Example

The following XML output shows metadata returned in a GDO for a classical album. See the musicid\_gdo\_navigation sample application for an example showing how to access classical metadata. Also see AUDIO\_WORK in the [Data Model](#).

```
<ALBUM>
  <PACKAGE_LANG ID="1" LANG="eng">English</PACKAGE_LANG>
```

```

<ARTIST>
  <NAME_OFFICIAL>
    <DISPLAY>Various Artists</DISPLAY>
  </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Various Artists</DISPLAY>
    </NAME_OFFICIAL>
  </CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
  <DISPLAY>Grieg: Piano Concerto, Peer Gynt Suites #1 &
2</DISPLAY>
  </TITLE_OFFICIAL>
  <YEAR>1989</YEAR>
  <GENRE_LEVEL1>Classical</GENRE_LEVEL1>
  <GENRE_LEVEL2>Romantic Era</GENRE_LEVEL2>
  <LABEL>LaserLight</LABEL>
  <TOTAL_IN_SET>1</TOTAL_IN_SET>
  <DISC_IN_SET>1</DISC_IN_SET>
  <TRACK_COUNT>3</TRACK_COUNT>
  <COMPILATION>Y</COMPILATION>
  <TRACK_ORD="1">
  <TRACK_NUM>1</TRACK_NUM>
  <ARTIST>
    <NAME_OFFICIAL>
      <DISPLAY>János Sándor: Budapest Philharmonic Orchestra</DISPLAY>
    </NAME_OFFICIAL>
    <CONTRIBUTOR>
      <NAME_OFFICIAL>
        <DISPLAY>János Sándor: Budapest Philharmonic
Orchestra</DISPLAY>
      </NAME_OFFICIAL>
      <ORIGIN_LEVEL1>Eastern Europe</ORIGIN_LEVEL1>
      <ORIGIN_LEVEL2>Hungary</ORIGIN_LEVEL2>
      <ORIGIN_LEVEL3>Hungary</ORIGIN_LEVEL3>
      <ORIGIN_LEVEL4>Hungary</ORIGIN_LEVEL4>
      <ERA_LEVEL1>2000's</ERA_LEVEL1>
      <ERA_LEVEL2>2000's</ERA_LEVEL2>
      <ERA_LEVEL3>2000's</ERA_LEVEL3>
      <TYPE_LEVEL1>Mixed</TYPE_LEVEL1>
      <TYPE_LEVEL2>Mixed Group</TYPE_LEVEL2>
    </CONTRIBUTOR>
  </ARTIST>
  <TITLE_OFFICIAL>

```

```

    <DISPLAY>Grieg: Piano Concerto In A Minor, Op. 16</DISPLAY>
  </TITLE_OFFICIAL>
  <GENRE_LEVEL1>Classical</GENRE_LEVEL1>
  <GENRE_LEVEL2>Other Classical</GENRE_LEVEL2>
</TRACK>
<TRACK_ORD="2">
<TRACK_NUM>2</TRACK_NUM>
<ARTIST>
  <NAME_OFFICIAL>
    <DISPLAY>Yuri Ahronovitch: Vienna Symphony Orchestra</DISPLAY>
  </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Yuri Ahronovitch: Vienna Symphony Orchestra</DISPLAY>
    </NAME_OFFICIAL>
    <ORIGIN_LEVEL1>Western Europe</ORIGIN_LEVEL1>
    <ORIGIN_LEVEL2>Austria</ORIGIN_LEVEL2>
    <ORIGIN_LEVEL3>Vienna</ORIGIN_LEVEL3>
    <ORIGIN_LEVEL4>Vienna</ORIGIN_LEVEL4>
    <ERA_LEVEL1>1990's</ERA_LEVEL1>
    <ERA_LEVEL2>1990's</ERA_LEVEL2>
    <ERA_LEVEL3>1990's</ERA_LEVEL3>
    <TYPE_LEVEL1>Mixed</TYPE_LEVEL1>
    <TYPE_LEVEL2>Mixed Group</TYPE_LEVEL2>
  </CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
  <DISPLAY>Grieg: Peer Gynt Suite #1, Op. 46</DISPLAY>
</TITLE_OFFICIAL>
<GENRE_LEVEL1>Classical</GENRE_LEVEL1>
<GENRE_LEVEL2>Other Classical</GENRE_LEVEL2>
</TRACK>
<TRACK_ORD="3">
<TRACK_NUM>3</TRACK_NUM>
<ARTIST>
  <NAME_OFFICIAL>
    <DISPLAY>Daniel Gerard; Peter Wohler: Berlin Radio Symphony
Orchestra</DISPLAY>
  </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Daniel Gerard; Peter Wohler: Berlin Radio Symphony
Orchestra</DISPLAY>
    </NAME_OFFICIAL>
    <ORIGIN_LEVEL1>Western Europe</ORIGIN_LEVEL1>

```

```
<ORIGIN_LEVEL2>Germany</ORIGIN_LEVEL2>
<ORIGIN_LEVEL3>Berlin</ORIGIN_LEVEL3>
<ORIGIN_LEVEL4>Berlin</ORIGIN_LEVEL4>
<ERA_LEVEL1>1990&apos;s</ERA_LEVEL1>
<ERA_LEVEL2>Early 90&apos;s</ERA_LEVEL2>
<ERA_LEVEL3>Early 90&apos;s</ERA_LEVEL3>
<TYPE_LEVEL1>Mixed</TYPE_LEVEL1>
<TYPE_LEVEL2>Mixed Group</TYPE_LEVEL2>
</CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
  <DISPLAY>Grieg: Peer Gynt Suite #2, Op. 55</DISPLAY>
</TITLE_OFFICIAL>
<GENRE_LEVEL1>Classical</GENRE_LEVEL1>
<GENRE_LEVEL2>Other Classical</GENRE_LEVEL2>
</TRACK>
</ALBUM>
```

### 3.11.9.2 Accessing External IDs

GNSDK can match identified media with External IDs (XIDs). These are third-party identifiers provided by preferred Gracenote partners that allow applications to match identified media to merchandise IDs in online stores and other services - facilitating transactions by helping connect queries directly to commerce. Gracenote has several preferred partnerships and pre-matches partner XIDs to Gracenote media IDs. Entitled applications can retrieve these IDs and present them to users.

XIDs are only available using GDO keys. So for albums for example, you set `ENABLE_LINK`, call `find_albums()`, then the resultant album GDO will have the XIDs in it.

To access XIDs for an Album:

1. Initialize manager and music modules.
2. Pass in a source GDO.
3. Call `gnsdk_*_query_option_Set()` with the option `OPTION_ENABLE_EXTERNAL_ID_DATA`.



4. Call `gnsdk_*_query_find_albums()` and navigate through the returned Album GDO for possible XIDs.
5. Get the External IDs using `gnsdk_manager_gdo_value_get()` and `GNSDK_GDO_VALUE_EXTERNALID_VALUE`.
6. Get the External IDs data using `gnsdk_manager_gdo_value_get()` and `GNSDK_GDO_VALUE_EXTERNALID_DATA`.

The following table lists options for `gnsdk_*_query_option_Set()` to access XIDs (External IDs).

To Retrieve	Function	Use
XIDs	<code>gnsdk_*_query_option_set()</code>	<code>GNSDK_*_OPTION_ENABLE_EXTERNAL_ID_DATA</code> query key.

where \* is MUSICID or MUSICIDFILE.

### Example

```
...
if (GNSDK_SUCCESS == error)
{
    gnsdk_uint32_t count = 0;
    gnsdk_uint32_t i = 0;
    gnsdk_gdo_handle_t xid_gdo = GNSDK_NULL;
    error = gnsdk_manager_gdo_child_count(album_gdo, GNSDK_GDO_CHILD_
EXTERNAL_ID, &count);
    if (GNSDK_SUCCESS == error)
    {
        printf( "    We have %d xid child\n", count );
        for (i = 1; i &lt;= count; i++)
        {
            error = gnsdk_manager_gdo_child_get( album_gdo, GNSDK_GDO_CHILD_
EXTERNAL_ID, i, &xid_gdo );
            if (GNSDK_SUCCESS == error)
            {
                error = gnsdk_manager_gdo_value_get( xid_gdo, GNSDK_GDO_VALUE_
EXTERNALID_VALUE, 1, &value );
                if (GNSDK_SUCCESS == error)
                {
                    printf( "    xid_value: %s\n", value );
                }
            }
        }
    }
}
```

```
    }
    if (GNSDKERR_NotFound == GNSDKERR_ERROR_CODE(error))
    {
        /*These fields are optional so it is totally fine if they
are not found at all*/
        error = GNSDK_SUCCESS;
    }
    error = gnsdk_manager_gdo_value_get( xid_gdo, GNSDK_GDO_VALUE_
EXTERNALID_SOURCE, 1, &value );
    if (GNSDK_SUCCESS == error)
    {
        printf( "    xid_source: %s\n", value );
    }
    if (GNSDKERR_NotFound == GNSDKERR_ERROR_CODE(error))
    {
        /*These fields are optional so it is totally fine if they
are not found at all*/
        error = GNSDK_SUCCESS;
    }
    error = gnsdk_manager_gdo_value_get( xid_gdo, GNSDK_GDO_VALUE_
EXTERNALID_TYPE, 1, &value );
    if (GNSDK_SUCCESS == error)
    {
        printf( "    xid_type: %s\n", value );
    }
    if (GNSDKERR_NotFound == GNSDKERR_ERROR_CODE(error))
    {
        /*These fields are optional so it is totally fine if they
are not found at all*/
        error = GNSDK_SUCCESS;
    }
    error = gnsdk_manager_gdo_value_get( xid_gdo, GNSDK_GDO_VALUE_
EXTERNALID_DATA, 1, &value );
    if (GNSDK_SUCCESS == error)
    {
        printf( "    xid_data: %s\n", value );
    }
    if (GNSDKERR_NotFound == GNSDKERR_ERROR_CODE(error))
    {
        /*These fields are optional so it is totally fine if they are
not found at all*/
        error = GNSDK_SUCCESS;
    }
}
gnsdk_manager_gdo_release(xid_gdo);
```

```
}  
}  
}
```

### 3.11.9.3 Accessing Collaborative Artists Metadata

Some songs are collaborations between two or more artists. For example, the Santana album “Supernatural” contains a number of collaborations, including:

- “Santana featuring Rob Thomas”
- “Santana featuring Dave Matthews”

You might want to take different actions in your application, depending on whether a song is by a single artist or a collaboration. The following topic shows how to navigate collaborations, and how to determine whether text-based lookup results contain collaborative artists. For information about best practices for working with collaborations, see *["Collaborative Artists Best Practices" on page 207](#)*.

#### Navigating Collaborations

In GNSDK, artists are represented by contributor GDOs. A contributor GDO can represent a single artist or a collaboration. To determine whether a contributor GDO represents a collaboration, check to see if it has a contributor child. If it does, the GDO represents a collaboration and the contributor child is the primary collaborator.

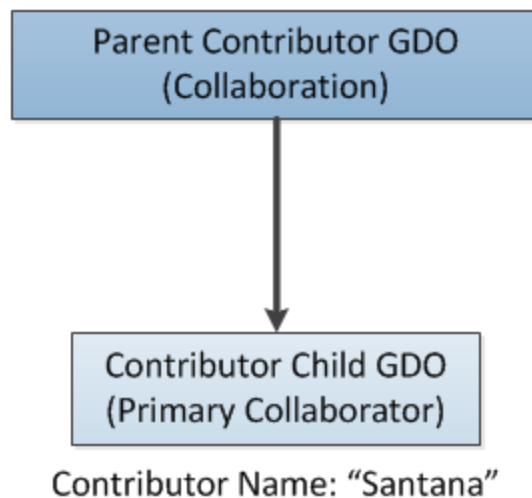
The following example checks to see whether the contributor GDO has a contributor child, and if so, it retrieves the primary collaborator GDOs:

```
error = gnsdk_manager_gdo_child_count(contributor_gdo,  
                                     GNSDK_GDO_CHILD_CONTRIBUTOR,&collab_count);  
/* The number of collaborators will either be 0 or 1 depending on  
 * whether or not this contributor is a collaboration.  
 */  
if (collab_count == 1)  
{  
    /* This contributor is a collaboration  
     * Retrieve the primary collaborator  
     */  
}
```

```
error = gnsdk_manager_gdo_child_get(contributor_gdo,  
    GNSDK_GDO_CHILD_CONTRIBUTOR, collab_count, &collaborator_gdo);  
if (!error)  
{  
    /* Navigate the collaborator GDO */  
    ...  
}  
}
```

In addition, the parent contributor GDO contains the collaboration name. As an example, if you have a contributor GDO that represents the collaboration “Santana featuring Rob Thomas,” it would have a contributor child GDO. The child GDO has the primary collaborator name “Santana,” and the parent GDO has the collaborator name “Santana featuring Rob Thomas.”

Contributor Name: “Santana feat. Rob Thomas”



### Working with Collaborative Artists in Text-Lookup Results

When you perform a text-based lookup using `gnsdk_musicid_query_find_matches()`, one of the possible results is a contributor GDO. This section discusses how to determine whether these contributor lookup results contain collaborative artists.

### *Using the GNSDK\_GDO\_VALUE\_COLLABORATOR\_RESULT Key*

If your input artist string represents a collaboration, but that collaboration is not found, GNSDK attempts to find the primary collaborator instead. If it is able to find the primary collaborator, it returns results for that collaborator, and the GNSDK\_GDO\_VALUE\_COLLABORATOR\_RESULT key is set to TRUE. This indicates that your input is a collaboration, but you only matched the primary collaborator. This allows you to distinguish this case from a situation in which the input was a single artist, and a single artist was identified.

For the purposes of this discussion, assume that you've done a text-based lookup that included an artist name as input, and you have gotten a contributor GDO back. The input text might be a single artist, such as "Santana," or it might be a collaboration, such as "Santana featuring Rob Thomas."

To determine whether the lookup results contain collaborative artists, use the following general steps:

1. Check to see whether the contributor GDO has a contributor child. If it does, the result is a collaboration and should match your input artist name ("Santana featuring Rob Thomas"). For more information, see *"Navigating Collaborations" on page 218*.
2. If the contributor GDO does not have children, use the GNSDK\_GDO\_VALUE\_COLLABORATOR\_RESULT key to distinguish between the following two cases:
  - If the key is FALSE, the result is a single artist that matched your input text ("Santana").
  - If the key is TRUE, your input is a collaboration ("Santana featuring Unknown Artist"), but you only matched the primary collaborator, so the single artist object is returned ("Santana"). In this case, you'll want to be careful not to overwrite the users' input collaboration name with the returned collaborator name.

### *Collaborative Artist Input Scenarios*

The following sections show the results for different inputs and provide more information about how to use the results in your application.

#### 3.11.9.4 Input Collaboration Found

The following table shows a case where the input string matches a collaboration in the database.

Example Input Text	"Santana featuring Rob Thomas"
GDO Results	A contributor GDO that has a child contributor GDO; the parent GDO has the collaboration name "Santana featuring Rob Thomas," and the child GDO has the primary collaborator name "Santana."

#### 3.11.9.5 Input Collaboration Not Found

The following table shows a case where the input string does not match a collaboration in the database, but the input string is identified as a collaboration. In this case, the lookup returns the primary collaborator, and the GNSDK\_GDO\_VALUE\_COLLABORATOR\_RESULT flag is set to TRUE, which indicates that the input is a collaboration, but the lookup only returned the primary collaborator.

Example Input Text	"Santana featuring Unknown Artist"
GDO Results	One contributor GDO with artist name "Santana"
GNSDK_GDO_VALUE_COLLABORATOR_RESULT Value	TRUE



**Note:** If this track is being played, Gracenote recommends that you display the original input text in your interface, but use the returned collaborator to generate playlists for the input track.

### 3.11.9.6 Input is Not a Collaboration

The following table shows a case where the input string is not a collaboration, and a single artist match is found.

Example Input Text	"Santana"
GDO Results	One contributor GDO with artist name "Santana"
GNSDK_GDO_VALUE_COLLABORATOR_RESULT Value	FALSE

For more information, see *"Identifying Music Using Text" on page 167*.

### 3.11.9.7 Improving Matches Using Both CD TOC and Fingerprints

To help improve the accuracy of the results returned from Gracenote Service, you can use both a TOC and a fingerprint in a query. The use of a fingerprint for additional identification criteria can increase the number of single- and multi-exact matches and decreasing the number of fuzzy match results. This query method is especially useful for Albums that contain few (four or less) Tracks, such as CD singles. With this method:

- You do not need to set the TOC and fingerprint functions in a specific order on the query handle. Setting the TOC first and the fingerprint second, or vice versa, will work.
- The fingerprint can be of either metadata type: CMX or GNFPX.
- You can use existing fingerprint data, or generate fingerprint data. See the examples below.
- When the fingerprint metadata does not aid the match, Gracenote Service disregards it, and returns a result (or results) that match only the input TOC.
- Using this query method generally results in a longer query processing time to identify the additional fingerprint; on average, about 15 seconds per Track.

- You can alternately perform this query method using `gnsdk_musicid_query_add_toc_offset()`.

If experiencing issues when performing cumulative queries, check that the logs are recording both a TOC and a fingerprint as inputs for the specific query.

### Using a TOC and an Existing Fingerprint

Use a TOC with an existing fingerprint that is accessed using `gnsdk_musicid_query_set_fp_data()`. In this case, the fingerprint does not have to be the first Track of the Album. A calling sequence example is:

1. `gnsdk_musicid_query_set_toc_string()`
2. `gnsdk_musicid_query_set_fp_data()`
3. `gnsdk_musicid_query_find_albums()`

### Using a TOC and a Generated Fingerprint

Use a TOC with fingerprint that is generated using the `gnsdk_musicid_query_fingerprint_*` functions. Calling `gnsdk_musicid_query_fingerprint_end()` is optional, depending on whether the `pb_complete` parameter of `gnsdk_musicid_query_fingerprint_write()` receives enough audio data. Refer to these functions' Remarks for more information. A calling sequence example is:

1. `gnsdk_musicid_query_set_toc_string()`
2. `gnsdk_musicid_query_fingerprint_begin()`
3. `gnsdk_musicid_query_fingerprint_write()`
4. (Optional)`gnsdk_musicid_query_fingerprint_end()`
5. `gnsdk_musicid_query_find_albums()`

### 3.11.9.8 Accessing Enriched Content using Asset Fetch

You can access enriched content through the Manager Asset Fetch APIs, `gnsdk_manager_asset_fetch2()`, or through the Link Module.

Enriched data includes:



- Album cover art
- Artist images
- Genre art (only available via the Link module)

## Using the Asset Fetch API

The Asset Fetch APIs are the preferred way to access enriched content.

To retrieve cover art or an artist image, you need an album or contributor GDO from media that has been identified using GNSDK recognition features. For example, you might have identified an album using a CD TOC or a text-based lookup. For more information, see [Identifying Music](#).

Once you have an appropriate content GDO, you follow these steps to fetch the asset data:

1. From the album or contributor, retrieve the content GDO based on content type with `gnsdk_manager_gdo_child_get()`.
2. From the content GDO, find the asset GDO based on the desired image size with `gnsdk_manager_gdo_child_get()`.
3. From the asset GDO, use an asset URL key to get the URL with `gnsdk_manager_gdo_value_get()`.
4. From the URL, retrieve the asset data with `gnsdk_manager_asset_fetch2()`.

These steps are shown in the following snippet:

```
/*Note: Error checking removed for brevity. Always check GNSDK return codes. */

/*Get an album GDO from some source */
gnsdk_manager_gdo_child_get(list_gdo, GNSDK_GDO_CHILD_ALBUM,i,&album_gdo);

/*From the album, retrieve the content GDO based on content type */
gnsdk_manager_gdo_child_get(album_gdo, GNSDK_GDO_CHILD_CONTENT_IMAGECOVER, i, &content_gdo);

/*From the content, retrieve the asset GDO based on the desired image
```

```
size */
gnsdk_manager_gdo_child_get(album_gdo, GNSDK_GDO_CHILD_ASSET_SIZE_
SMALL, i, &asset_gdo);

/*From the asset, retrieve the URL */
gnsdk_manager_gdo_value_get(asset_gdo, GNSDK_GDO_VALUE_ASSET_URL_
GNSDK, 1, &url);

/*From the URL, retrieve the asset data. You can retrieve an asset
either locally or online */
/*In this example, a flag (defined elsewhere) is used to determine
local or online lookup */
gnsdk_manager_asset_fetch2(url,
use_local ?
GNSDK_LOOKUP_MODE_LOCAL : GNSDK_LOOKUP_MODE_ONLINE_NOCACHE,
user_handle, GNSDK_NULL, GNSDK_NULL, &p_asset_data, &p_asset_data_size,
GNSDK_NULL);
```

### ***Example: Accessing Album Cover Art***

This example shows how to access album cover art. It does an album search and uses the above functions to find the album art.

Sample Application: musicid\_asset\_fetch/main.c

#### **3.11.9.9 Accessing Enriched Content using Link**

You can access enriched content through the Manager Asset Fetch APIs, `gnsdk_manager_asset_fetch2()`, or through the Link Module.

Enriched data includes:

- Album cover art
- Artist images
- Genre art (only available via the Link module)

## Using the Link Module

You can use the [Link Module](#) to retrieve enriched content. Link requires the following define:

```
#define GNSDK_LINK 1
```

And making the following initializing API call with your SDK Manager handle:

```
gnsdk_link_initialize(sdkmgr_handle);
```

Before your program exits, it needs to make the following API call to release Link resources:

```
gnsdk_link_shutdown();
```

## Retrieving Cover Art and Artist Images

To retrieve cover art or an artist image, you need an album or contributor GDO from media that has been identified using GNSDK recognition features. For example, you might have identified an album using a CD TOC or a text-based lookup, or you might have identified a contributor using a text-based lookup. For more information, see [Identifying Music](#).

Once you have a GDO, you can use the Link APIs to retrieve enriched content immediately, or you can serialize the GDO and deserialize it later, when you are ready to retrieve content.

In general, to access enriched image content:

1. Initialize the Manager, Music, and Link modules.
2. Create a Link query.
3. Set the GDO for the query.
4. Set the desired image size.
5. Retrieve the image, specifying the image type.
6. When finished using the image, free the image.



**Note:** Images that are retrieved with an online query can be cached locally, depending on the lookup mode option, and whether caching has been enabled. For more information, see [Setting Local and Online Lookup Modes](#) and [Using SQLite for Storage and Caching](#).

Use the following function to create a Link query:

```
gnsdk_link_query_create(user_handle, GNSDK_NULL, GNSDK_NULL, &query_handle);
```

In this example, the callback parameters have been set to GNSDK\_NULL, but you can use them to provide a callback function that will give status updates for the query, if you wish.

Next, set the GDO for the query. The GDO should be one that you've retrieved using a GNSDK recognition feature:

```
gnsdk_link_query_set_gdo(query_handle, input_gdo);
```

Set the desired image size. For a list of available image sizes, see [Image Formats and Dimensions](#).

```
gnsdk_link_query_option_set(query_handle, GNSDK_LINK_OPTION_KEY_IMAGE_SIZE,  
    GNSDK_LINK_OPTION_VALUE_IMAGE_SIZE_170);
```

Finally, call the `gnsdk_link_query_content_retrieve()` function to retrieve the image. You must specify the type of image to retrieve, by using one of the following constants:

- `gnsdk_link_content_cover_art` for album cover art
- `gnsdk_link_content_genre_art` for genre art
- `gnsdk_link_content_image_artist` for an artist image

```
gnsdk_link_query_content_retrieve(  
    query_handle,  
    gnsdk_link_content_cover_art,  
    1,  
    &data_type,
```

```
&buffer,  
&buffer_size);
```

When finished using the image, free the image buffer:

```
gnsdk_link_query_content_free(buffer);
```

## Retrieving Genre Art

Gracenote recommends retrieving and displaying genre art in the following cases:

- You've identified an album or contributor, but no cover art or artist image is found.
- You are not able to identify an album or contributor, but you are still able to identify the genre of the album or contributor.

### *Retrieving Genre Art with a GDO*

If you've identified an album or contributor, but no cover art or artist image is found, you can retrieve genre art by using the identified GDO. Create a Link query, as described in [Retrieving Cover Art and Artist Images](#), and set the image type to `gnsdk_link_content_genre_art`.

### *Retrieving Genre Art without a GDO*

If you are not able to identify an album or contributor, you might still be able to identify the genre by using the ID3 tags of the MP3 file. Create a Link query, as described in [Retrieving Cover Art and Artist Images](#), and set the image type to `gnsdk_link_content_genre_art`. However, instead of setting the GDO for the query, use the genre string to get the list element handle for the genre.

For example, use the following function to retrieve the list element handle:

```
gnsdk_manager_list_get_element_by_string(genre_list_handle, genre_  
string,  
    &element_handle);
```

Then set the list element handle for the Link query:

```
gnsdk_link_query_set_list_element(query_handle, element_handle);
```

## Setting Image Size Retrieval Order

You can specify an order in which GNSDK will retrieve image sizes, by setting the image size option multiple times. The first size that you set will be the first image size that GNSDK tries to retrieve. If an image of that size isn't found, the second size will be the next size that GNSDK tries to retrieve, and so on. For example, the following code calls the `gnsdk_link_query_option_set()` function three times, so that GNSDK will try to retrieve images in the following order: 300, 450, 170:

```
gnsdk_link_query_option_set(query_handle, GNSDK_LINK_OPTION_KEY_IMAGE_SIZE,
    GNSDK_LINK_OPTION_VALUE_IMAGE_SIZE_300);

gnsdk_link_query_option_set(query_handle, GNSDK_LINK_OPTION_KEY_IMAGE_SIZE,
    GNSDK_LINK_OPTION_VALUE_IMAGE_SIZE_450);

gnsdk_link_query_option_set(query_handle, GNSDK_LINK_OPTION_KEY_IMAGE_SIZE,
    GNSDK_LINK_OPTION_VALUE_IMAGE_SIZE_170);
```

The images sizes that you set will apply to the query handle for the life of the handle. If you want to retrieve images of a different size, you'll need to call the `gnsdk_link_query_options_clear()` function to clear the options on the handle. Then you can reset the image sizes before retrieving additional images.

### ***Example: Accessing Album Cover Art***

This example shows how to access album cover art. It does a text search and finds images based on gdo type (album or contributor). It also finds an image based on genre.

Sample Application: `music_image_fetch/main.c`

### 3.11.9.10 Using Enriched Content URLs

There may be cases when you wish to retrieve metadata, descriptors, or external identifiers as quickly as possible, while lazy-loading enriched content such as cover art in the background.

To achieve this, the GNSDK returns URL pointers to the enriched content, giving you greater control over when you load the additional content. These URLs are returned in every result response object if your application has acquired the necessary licensing for this data.



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

To retrieve a URL, use function `gnsdk_manager_gdo_value_get()` with a URL retrieval option as shown in the example below:

```
error = gnsdk_manager_gdo_value_get(channel_gdo, GNSDK_GDO_VALUE_URL_TVCHANNEL_IMAGE_220, 1, &channelImg);
```

The following table lists the available URL retrieval options.

Option	Value
GNSDK_GDO_VALUE_URL_GENRE_IMAGE_*	Retrieves the temporary URL for the music genre art image, based on image size. For example, GNSDK_GDO_VALUE_URL_GENRE_URL_75 retrieves the URL for a 75x75 pixel image. Sizes available are 75, 170, 300, 450, 720, and 1080.
GNSDK_GDO_VALUE_URL_IMAGE_*	Retrieves the temporary URL for the image. In addition to the sizes listed above for GNSDK_GDO_VALUE_URL_GENRE_IMAGE_*, you can specify 75 or larger, 1080 or smaller, and larger or smaller for the intermediate sizes. For example, you can specify GNSDK_GDO_VALUE_URL_IMAGE_300, GNSDK_GDO_VALUE_URL_IMAGE_300_OR_SMALLER, or GNSDK_GDO_VALUE_URL_IMAGE_300_OR_LARGER.

Option	Value
GNSDK_GDO_VALUE_URL_VIDEO_CONTRIBUTOR_IMAGE_*	Retrieves the temporary URL for the video contributor image. Sizes available are the same as for GNSDK_GDO_VALUE_URL_IMAGE_*.
GNSDK_GDO_VALUE_URL_MUSIC_CONTRIBUTOR_IMAGE_*	Retrieves the temporary URL for the music contributor page. Sizes available are the same as for GNSDK_GDO_VALUE_URL_IMAGE_*.

### 3.11.9.11 Music Metadata Reference Overview

The following metadata reference examples are intended to give the reader some idea of what is returned from MusicID Query APIs. Note that they do not contain every possible field that could be returned. For a complete reference, consult the GNSDK Data Model.

#### Album Metadata Example

The following is intended to give the reader some idea of the metadata returned in an album GDO.

```
<ALBUM>
  <PACKAGE_LANG ID="1" LANG="eng">English</PACKAGE_LANG>
  <ARTIST>
    <NAME_OFFICIAL>
      <DISPLAY>Nelly</DISPLAY>
    </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Nelly</DISPLAY>
    </NAME_OFFICIAL>
    <ORIGIN_LEVEL1>North America</ORIGIN_LEVEL1>
    <ORIGIN_LEVEL2>United States</ORIGIN_LEVEL2>
    <ORIGIN_LEVEL3>Missouri</ORIGIN_LEVEL3>
```



```

    <ORIGIN_LEVEL4>St. Louis</ORIGIN_LEVEL4>
    <ERA_LEVEL1>2000&apos;s</ERA_LEVEL1>
    <ERA_LEVEL2>2000&apos;s</ERA_LEVEL2>
    <ERA_LEVEL3>2000&apos;s</ERA_LEVEL3>
    <TYPE_LEVEL1>Male</TYPE_LEVEL1>
    <TYPE_LEVEL2>Male</TYPE_LEVEL2>
  </CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
<DISPLAY>Nellyville</DISPLAY>
</TITLE_OFFICIAL>
<YEAR>2002</YEAR>
<GENRE_LEVEL1>Urban</GENRE_LEVEL1>
<GENRE_LEVEL2>Western Hip-Hop/Rap</GENRE_LEVEL2>
<LABEL>Universal</LABEL>
<TOTAL_IN_SET>1</TOTAL_IN_SET>
<DISC_IN_SET>1</DISC_IN_SET>
<TRACK_COUNT>19</TRACK_COUNT>
<TRACK_ORD="1">
<TRACK_NUM>1</TRACK_NUM>
<TITLE_OFFICIAL>
  <DISPLAY>Nellyville</DISPLAY>
</TITLE_OFFICIAL>
</TRACK>
<TRACK_ORD="2">
<TRACK_NUM>2</TRACK_NUM>
<ARTIST>
  <NAME_OFFICIAL>
    <DISPLAY>Nelly Feat. Cedric The Entertainer & La La</DISPLAY>
  </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Nelly Feat. Cedric The Entertainer & La
La</DISPLAY>
    </NAME_OFFICIAL>
  </CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
  <DISPLAY>Gettin It Started</DISPLAY>
</TITLE_OFFICIAL>
</TRACK>
<TRACK_ORD="3">
<TRACK_NUM>3</TRACK_NUM>
<TITLE_OFFICIAL>
  <DISPLAY>Hot In Herre</DISPLAY>

```

```
</TITLE_OFFICIAL>
<YEAR>2002</YEAR>
</TRACK>
<TRACK ORD="4">
<TRACK_NUM>4</TRACK_NUM>
<TITLE_OFFICIAL>
  <DISPLAY>Dem Boyz</DISPLAY>
</TITLE_OFFICIAL>
</TRACK>
<TRACK ORD="5">
<TRACK_NUM>5</TRACK_NUM>
<TITLE_OFFICIAL>
  <DISPLAY>Oh Nelly</DISPLAY>
</TITLE_OFFICIAL>
</TRACK>
<TRACK ORD="6">
<TRACK_NUM>6</TRACK_NUM>
<TITLE_OFFICIAL>
  <DISPLAY>Pimp Juice</DISPLAY>
</TITLE_OFFICIAL>
</TRACK>
<TRACK ORD="7">
<TRACK_NUM>7</TRACK_NUM>
<TITLE_OFFICIAL>
  <DISPLAY>Air Force Ones</DISPLAY>
</TITLE_OFFICIAL>
<YEAR>2002</YEAR>
</TRACK>
<TRACK ORD="8">
<TRACK_NUM>8</TRACK_NUM>
<ARTIST>
  <NAME_OFFICIAL>
    <DISPLAY>Nelly Feat. Cedric The Entertainer & La La</DISPLAY>
  </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Nelly Feat. Cedric The Entertainer & La
La</DISPLAY>
    </NAME_OFFICIAL>
  </CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
  <DISPLAY>In The Store</DISPLAY>
</TITLE_OFFICIAL>
</TRACK>
```

```
<TRACK ORD="9">
<TRACK_NUM>9</TRACK_NUM>
<ARTIST>
  <NAME_OFFICIAL>
    <DISPLAY>Nelly Feat. King Jacob</DISPLAY>
  </NAME_OFFICIAL>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Nelly Feat. King Jacob</DISPLAY>
    </NAME_OFFICIAL>
    <ORIGIN_LEVEL1>North America</ORIGIN_LEVEL1>
    <ORIGIN_LEVEL2>United States</ORIGIN_LEVEL2>
    <ORIGIN_LEVEL3>Missouri</ORIGIN_LEVEL3>
    <ORIGIN_LEVEL4>St. Louis</ORIGIN_LEVEL4>
    <ERA_LEVEL1>2000&apos;s</ERA_LEVEL1>
    <ERA_LEVEL2>Early 2000&apos;s</ERA_LEVEL2>
    <ERA_LEVEL3>Early 2000&apos;s</ERA_LEVEL3>
    <TYPE_LEVEL1>Male</TYPE_LEVEL1>
    <TYPE_LEVEL2>Male Duo</TYPE_LEVEL2>
  </CONTRIBUTOR>
</ARTIST>
<TITLE_OFFICIAL>
  <DISPLAY>On The Grind</DISPLAY>
</TITLE_OFFICIAL>
<GENRE_LEVEL1>Urban</GENRE_LEVEL1>
<GENRE_LEVEL2>Western Hip-Hop/Rap</GENRE_LEVEL2>
</TRACK>

... More Tracks

</ALBUM>
```

### 3.11.9.12 Generating a Playlist

Generating a Playlist involves the following general steps.

1. Create a new Playlist collection summary.
2. Populate the collection summary with unique identifiers and GDOs.
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, using the More Like This function with a seed.
6. Access and display playlist results.

To generate a custom playlist, use the Playlist Definition Language. For more information, see *"Playlist PDL Specification" on page 292*.

### Creating a Collection Summary

To create a collection summary, call the following function:

```
gnsdk_playlist_collection_create("sample_collection", &playlist_
collection);
```

This creates a new and empty collection summary. The function returns a pointer to a collection handle, which can be used in subsequent calls.



**Note:** Each new collection summary that you want to save must have a unique name, or the previously saved summary with the same name will be overwritten.

### Populating a Collection Summary

The main task in building a Playlist collection summary is to provide all possible data to the Playlist for each specific media item. The API to provide data for a collection summary is `gnsdk_playlist_collection_add_gdo()`. This API takes a media identifier (any unique string determined by the application) and a GDO to acquire data from. The GDO should come from a recognition event from the other GNSDK modules (such as MusicID or MusicID-File).

### *Enabling Playlist Attributes*

When creating a MusicID or MusicID-File query to populate a playlist, you must set the following query options with the appropriate function (for example, `gnsdk_`

musicid\_query\_option\_set() for the MusicID module) to ensure that the appropriate Playlist attributes are returned (depending on the type of query):

- GNSDK\_MUSICID\_OPTION\_ENABLE\_SONIC\_DATA or GNSDK\_MUSICIDFILE\_OPTION\_ENABLE\_SONIC\_DATA
- GNSDK\_MUSICID\_OPTION\_ENABLE\_PLAYLIST or GNSDK\_MUSICIDFILE\_OPTION\_ENABLE\_PLAYLIST

### ***Adding Data to a Collection Summary***

You can add an identifier to a collection using `gnsdk_playlist_collection_add_ident()`. This function creates an empty entry in the collection summary for the given identifier. An application provides an identifier that allows the application to identify referenced physical media. Identifiers are not interpreted by Playlist, but are only returned to the application in Playlist results. Identifiers are application dependent. For example, a desktop application might use a full path to a file name, while an online application might use a database key. Once you have added an identifier, an application can add data to it using the other `gnsdk_playlist_collection_add_*`() APIs.

For example, after retrieving GDOs with a query, use the `gnsdk_playlist_collection_add_gdo()` function to add the GDOs to the collection summary. As shown below, you might call the function with an album or track GDO (or both), depending on your use case:

```
gnsdk_playlist_collection_add_gdo(playlist_collection, unique_ident,  
album_gdo);  
  
gnsdk_playlist_collection_add_gdo(playlist_collection, unique_ident,  
track_gdo);
```



**Note:** You can add multiple GDOs for the same identifier by calling the `gnsdk_playlist_collection_add_gdo()` API multiple times with the same identifier value. The data gathered from all the provided GDOs is stored for the given identifier. The identifier is a user generated value that should allow the identification of individual tracks (for example, path/filename) or an ID that can be externally looked up.

When adding an album GDO to Playlist that resulted from a CD TOC lookup, Gracenote recommends adding all tracks from the album to Playlist individually. To do this, call `gnsdk_playlist_collection_add_gdo()` twice for each track on the album. The first call should pass the album GDO for the identifier (to allow Playlist to gather album data), and the second call should pass the respective track GDO from the album using the same identifier used in the first call. This will ensure Playlist adds full album and track data to the collection summary for each track on the album, which supports querying the Playlist Collection with both track and album level attributes.

### ***How Playlist Gathers Data***

When given an album GDO, Playlist gathers any necessary album data and then traverses to the matched track and gathers the necessary data from that track. This data is stored for the given identifier. If no matched track is part of the album GDO, no track data is gathered. Playlist also gathers data from both the album and track contributors as detailed below.

When given a track GDO, 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 GDO (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.

For all other GDOs, Playlist attempts to gather the necessary data, but no other specific traversals are done automatically.

### ***Adding List Elements to Collection Summaries***

You can use genre and other list data (origin, era, and tempo) to add media items to playlists, by matching by string and adding the result to a collection summary. To search for list elements by string, use the `gnsdk_manager_list_get_element_by_string()` function. This API returns a `gnsdk_list_element_handle_t`, which can be passed to the `gnsdk_playlist_collection_add_list_element()` function to add the value to a collection summary.



**Note:** When searching for a genre list element, the SDK can match many different genre variations in many different languages, regardless of the language specified when loading a locale.

## Working with Local Storage

You can store and manage collection summaries in local storage. To store a collection summary, use the following function:

```
gnsdk_playlist_storage_store_collection(playlist_collection);
```

To retrieve a collection summary from local storage, use the `gnsdk_playlist_storage_count_collections()` and `gnsdk_playlist_storage_enum_collections()` functions to retrieve all collections.

Before Playlist can use a collection summary that has been retrieved from storage, it must be loaded into memory:

```
gnsdk_playlist_storage_load_collection(collection_name, &playlist_collection);
```

## Generating a Playlist Using More Like This

You can streamline your Playlist implementation by using the `gnsdk_playlist_generate_morelikethis()` function, which uses the "More Like This" algorithm to generate optimal playlist results and eliminates the need to create and validate Playlist Definition Language statements.

You can set the following options when generating a More Like This Playlist, by using the `gnsdk_playlist_morelikethis_option_set()` function:

Option	Description
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_TRACKS	The maximum number of track results returned. Default is 25.
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ARTIST	The maximum number of results per artist returned. Default is 2.
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ALBUM	The maximum number of results per album returned; the value for this key must evaluate to a number greater than 0. Default is 1.
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_RANDOMSEED	The randomization seed value used in calculating a More Like This playlist. The value for this key must evaluate to a number greater than 0 (the recommended range is 1 - 100, but you can use any number from 1 to 4294967295. This number will be the seed for the randomization. You can re-create a playlist by choosing the same number for the randomization seed; choosing different numbers will create different playlists. Default is 0.

The following example demonstrates setting the More Like This options:

```

/*
 * Change the possible result set to contain a maximum of 30 tracks
 */
gnsdk_playlist_morelikethis_option_set(
    playlist_collection,
    GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_TRACKS,
    "30"
);

/*
 * Change the possible result set to contain a maximum of 10 tracks
 * per artist
 */
gnsdk_playlist_morelikethis_option_set(

```



```
    playlist_collection,  
    GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ARTIST,  
    "0"  
);  
  
/*  
 * Change the possible result set to contain a maximum of 5 tracks per  
album  
 */  
gnsdk_playlist_morelikethis_option_set(  
    playlist_collection,  
    GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ALBUM,  
    "5"  
);  
  
/*  
 * Change the random result to be 1, so that there is no randomization  
 */  
gnsdk_playlist_morelikethis_option_set(  
    playlist_collection,  
    GNSDK_PLAYLIST_MORELIKETHIS_OPTION_RANDOMSEED,  
    "1"  
);
```

To generate a More Like This playlist, call the `gnsdk_playlist_generate_morelikethis()` function with a user handle, a playlist collection summary, and a seed GDO. A seed GDO can be any recognized media GDO, for example, the GDO of the track that is currently playing.

```
gnsdk_playlist_generate_morelikethis(  
    user_handle,  
    playlist_collection,  
    h_gdo_seed,  
    &playlist_results  
);
```

## Accessing Playlist Results

Once you have generated a playlist, you can iterate through the results using the `gnsdk_playlist_results_count()` and `gnsdk_playlist_results_enum()` functions. The results consist of a set of unique identifiers, which match the identifiers given to

Playlist during the population of the collection summary. The application must match these identifiers to the physical media that they reference.

### 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 the `gnsdk_playlist_collection_join()` function:

```
gnsdk_playlist_collection_join(collection_handle1, collection_
handle2);
```

Calling this function joins collection 2 into collection 1. The collection 1 handle now represents the joined collection, and can be used to generate the playlist.

You can also enumerate identifiers across joined collections by using the `gnsdk_playlist_collection_ident_count()` and `gnsdk_playlist_collection_ident_enum()` functions.

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 GDOs, 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 `gnsdk_playlist_collection_join_remove()` function.

## ***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.

## **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.

To synchronize a collection summary to physical media:

1. **Iterate the physical media** – Add all existing (current and new) unique identifiers using the `gnsdk_playlist_collection_sync_ident_add()` function, which Playlist collates.
2. **Process the collection** – Call `gnsdk_playlist_collection_sync_process()` 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 `gnsdk_playlist_collection_sync_ident_add()` function for each media item. For each media item, pass the unique identifier associated with the item to the function. 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 `gnsdk_playlist_collection_sync_ident_add()`, call `gnsdk_playlist_collection_sync_process()` to synchronize the collection summary's data. During processing, the callback function `gnsdk_playlist_update_callback_fn()` will be called for each difference between the physical media and the collection summary. So 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. For more information on adding items, see [Populating a Collection Summary](#). To remove items, use the `gnsdk_playlist_collection_ident_remove()` function.

### **Example: Implementing a Playlist**

This example demonstrates using the Playlist APIs to create More Like This and custom playlists.

Sample Application: `playlist/main.c`

Application steps:

1. Initialize the GNSDK, enable logging, initialize SQLite, and local lookup
2. Initialize the MusicID and Playlist modules, and set the location for the collection store
3. Get a User handle and load a Locale
4. Create a Playlist collection
5. Demonstrate PDL usage

## 6. Demonstrate More Like This usage

## 7. Cleanup resources and shutdown modules and GNDSK

### Sample output:

```
GNSDK Product Version   : 3.05.0.798    (built 2013-05-08 16:09-0700)

Currently stored collections :1
  Loading Collection 'sample_collection' from store
All Playlist Attributes:
  GN_AlbumName
  GN_ArtistName
  GN_ArtistType
  GN_Era
  GN_Genre
  GN_Origin
  GN_Mood
  GN_Tempo
Collection Attributes: 'sample_collection' (287 idents)
  GN_AlbumName
  GN_ArtistName
  GN_ArtistType
  GN_Era
  GN_Genre
  GN_Origin
  GN_Mood
  GN_Tempo

PDL 0: GENERATE PLAYLIST WHERE (GN_Genre LIKE 2929) > 0
Generated Playlist: 79
  1: 4_1 Collection Name:sample_collection
    GN_AlbumName:Come Away With Me
    GN_ArtistName:Norah Jones
    GN_Era:2002
    GN_Genre:Western Pop
    GN_Mood:Depressed / Lonely
    GN_Tempo:Slow

    ...

  79: 14_13 Collection Name:sample_collection
    GN_AlbumName:Supernatural
    GN_ArtistName:Santana
    GN_ArtistType:Male Group
    GN_Era:1970's
```

```
GN_Genre:70's Rock
GN_Origin:San Francisco
GN_Mood:Dark Groovy
GN_Tempo:Medium Fast

PDL 1: GENERATE PLAYLIST WHERE (GN_Genre LIKE 2929) > 300
Generated Playlist: 43
  1: 4_1 Collection Name:sample_collection
    GN_AlbumName:Come Away With Me
    GN_ArtistName:Norah Jones
    GN_Era:2002
    GN_Genre:Western Pop
    GN_Mood:Depressed / Lonely
    GN_Tempo:Slow

    ...

  43: 17_12 Collection Name:sample_collection
    GN_AlbumName:Breakaway
    GN_ArtistName:Kelly Clarkson
    GN_Era:2004
    GN_Genre:Western Pop
    GN_Mood:Dramatic Emotion
    GN_Tempo:Slow

PDL 2: GENERATE PLAYLIST WHERE GN_Genre = 2929
Generated Playlist: 0

PDL 3: GENERATE PLAYLIST WHERE GN_Genre = 2821
Generated Playlist: 0

PDL 4: GENERATE PLAYLIST WHERE (GN_Genre LIKE 2821) > 0
Generated Playlist: 71
  1: 3_26 Collection Name:sample_collection
    GN_AlbumName:1
    GN_ArtistName:The Beatles
    GN_Era:2000
    GN_Genre:60's Rock
    GN_Mood:Dramatic Emotion
    GN_Tempo:Medium

    ...

  71: 16_14 Collection Name:sample_collection
    GN_AlbumName:Paper Music
```

```
GN_ArtistName:Bobby McFerrin
GN_ArtistType:Mixed Group
GN_Era:Mid 90's
GN_Genre:Western Pop
GN_Origin:Minnesota
GN_Mood:Solemn / Spiritual
GN_Tempo:Very Slow
```

```
PDL 5: GENERATE PLAYLIST WHERE (GN_Genre LIKE 2821) > 300
Generated Playlist: 57
```

```
1: 15_13 Collection Name:sample_collection
GN_AlbumName:Supernatural
GN_ArtistName:Santana
GN_ArtistType:Male Group
GN_Era:Late 90's
GN_Genre:70's Rock
GN_Origin:San Francisco
GN_Mood:Dark Groovy
GN_Tempo:Medium Fast
```

```
...
```

```
57: 7_6 Collection Name:sample_collection
GN_AlbumName:All The Right Reasons
GN_ArtistName:Nickelback
GN_ArtistType:Male Group
GN_Era:2000's
GN_Genre:Alternative
GN_Origin:Alberta
GN_Mood:Powerful / Heroic
GN_Tempo:Fast
```

```
PDL 6: GENERATE PLAYLIST WHERE (GN_Genre LIKE SEED) > 300 LIMIT 20
RESULTS
```

```
Generated Playlist: 20
1: 0_13 Collection Name:sample_collection
GN_AlbumName:American Idiot
GN_ArtistName:Green Day
GN_ArtistType:Male Group
GN_Era:1990's
GN_Genre:Punk
GN_Origin:Northern Calif.
GN_Mood:Energetic Anxious
GN_Tempo:Medium Fast
```

```
...

20: 7_1 Collection Name:sample_collection
    GN_AlbumName:All The Right Reasons
    GN_ArtistName:Nickelback
    GN_ArtistType:Male Group
    GN_Era:2005
    GN_Genre:Alternative
    GN_Origin:Alberta
    GN_Mood:Heavy Brooding
    GN_Tempo:Fast

PDL 7: GENERATE PLAYLIST WHERE (GN_ArtistName LIKE 'Green Day') > 300
LIMIT 20 RESULTS, 2 PER GN_ArtistName;
Generated Playlist: 2
  1: 0_13 Collection Name:sample_collection
      GN_AlbumName:American Idiot
      GN_ArtistName:Green Day
      GN_ArtistType:Male Group
      GN_Era:1990's
      GN_Genre:Punk
      GN_Origin:Northern Calif.
      GN_Mood:Energetic Anxious
      GN_Tempo:Medium Fast
  2: 0_12 Collection Name:sample_collection
      GN_AlbumName:American Idiot
      GN_ArtistName:Green Day
      GN_ArtistType:Male Group
      GN_Era:1990's
      GN_Genre:Punk
      GN_Origin:Northern Calif.
      GN_Mood:Heavy Brooding
      GN_Tempo:Medium Fast

MoreLikeThis tests
MoreLikeThis Seed details:
    GN_AlbumName:American Idiot
    GN_ArtistName:Green Day
    GN_ArtistType:Male Group
    GN_Era:1990's
    GN_Genre:Punk
    GN_Origin:Northern Calif.
    GN_Mood:Energetic Yearning
    GN_Tempo:Medium Fast
```



MoreLikeThis with Default Options

```
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_TRACKS :25
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ARTIST :2
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ALBUM :1
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_RANDOMSEED :0
```

Generated Playlist: 2

1: 7\_9 Collection Name:sample\_collection

```
GN_AlbumName:All The Right Reasons
GN_ArtistName:Nickelback
GN_ArtistType:Male Group
GN_Era:2000's
GN_Genre:Alternative
GN_Origin:Alberta
GN_Mood:Energetic Anxious
GN_Tempo:Fast
```

2: 0\_5 Collection Name:sample\_collection

```
GN_AlbumName:American Idiot
GN_ArtistName:Green Day
GN_ArtistType:Male Group
GN_Era:1990's
GN_Genre:Punk
GN_Origin:Northern Calif.
GN_Mood:Energetic Yearning
GN_Tempo:Medium Fast
```

MoreLikeThis with Custom Options

```
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_TRACKS :30
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ARTIST :10
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_MAX_PER_ALBUM :5
GNSDK_PLAYLIST_MORELIKETHIS_OPTION_RANDOMSEED :1
```

Generated Playlist: 6

1: 0\_4 Collection Name:sample\_collection

```
GN_AlbumName:American Idiot
GN_ArtistName:Green Day
GN_ArtistType:Male Group
GN_Era:1990's
GN_Genre:Punk
GN_Origin:Northern Calif.
GN_Mood:Dreamy Brooding
GN_Tempo:Medium
```

...

```
6: 7_6 Collection Name:sample_collection
GN_AlbumName:All The Right Reasons
GN_ArtistName:Nickelback
GN_ArtistType:Male Group
GN_Era:2000's
GN_Genre:Alternative
GN_Origin:Alberta
GN_Mood:Powerful / Heroic
GN_Tempo:Fast
```

## Implementing Mood

The Mood APIs:

- Encapsulate Gracenote's Mood Editorial Content (mood layout and IDs).
- Simplify access to Mood results through x,y coordinates.
- Allow for multiple local and online data sources through Mood Providers.
- Enable pre-filtering of results using genre, origin, and era attributes.
- Support 5x5 or 10x10 Mood grids.
- Provide the ability to go from a cell of a 5x5 Mood grid to any of its expanded four Moods in a 10x10 grid.

Implementing Mood in an application involves the following steps:

1. Initializing the Mood module.
2. Enumerating the data sources using Mood Providers.
3. Creating and populating a Mood Presentation.
4. Filtering the results, if needed.

### ***Initializing the Mood Module***

Before using the Mood APIs, follow the usual steps to initialize GNSDK. The following GNSDK modules must be initialized:

- GNSDK Manager
- SQLite (for local caching)
- MusicID
- Playlist
- Mood

For more information on initializing the GNSDK modules, see “Initializing an Application.”

To initialize Mood, use the `gnsdk_moodgrid_initialize()` function.

```
error = gnsdk_moodgrid_initialize(sdkmgr_handle);
```



**Note:** If you are using MusicID to recognize music, you must enable Playlist and DSP data in your query. You must be entitled to use Playlist—if you are not, you won’t get an error, but Mood will return no results. Please contact your Gracenote representative for more information.

### ***Enumerating Data Sources using Mood Providers***

GNSDK automatically registers all local and online data sources available to Mood. For example, if you create a playlist collection using the Playlist API, GNSDK automatically registers that playlist as a data source available to Mood. These data sources are referred to as Providers. Mood is designed to work with multiple providers. You can iterate through the list of available Providers using the `gnsdk_moodgrid_provider_count()` and `gnsdk_moodgrid_provider_enum()` functions. For example, the following call returns a handle to the first Provider on the list (at index 0):

```
gnsdk_moodgrid_provider_enum(0, &provider);
```

You can use the handle to the Provider to retrieve the following information, by calling the `gnsdk_moodgrid_provider_get_data()` function:

- Name (GNSDK\_MOODGRID\_PROVIDER\_NAME key)
- Type (GNSDK\_MOODGRID\_PROVIDER\_TYPE key)

- Network Use (GNSDK\_MOODGRID\_PROVIDER\_NETWORK\_USE key)

### ***Creating and Populating a Mood Presentation***

Once you have a handle to a Provider, you can create and populate a Mood Presentation with Mood data. A Presentation is a data structure that represents the Mood, containing the mood name and playlist information associated with each cell of the grid.

To create a Mood Presentation, use the `gnsdk_moodgrid_presentation_create()` function, passing in the user handle, the type of the Mood, and the provider handle. The type can be one of the enumerated values in `gnsdk_moodgrid_presentation_type_t`: either a 5x5 or 10x10 grid. The function returns a handle to the Presentation:

```
gnsdk_moodgrid_presentation_create(user, type, NULL, NULL,
&presentation);
```

Each cell of the Presentation is populated with a mood name and associated playlist. You can iterate through the Presentation to retrieve this information from each cell. The following pseudo-code demonstrates this procedure:

```
gnsdk_moodgrid_presentation_type_dimension(type, &max_x, &max_y);
for (every grid cell [x,y] from [1,1] to [max_x, max_y])
{
    gnsdk_moodgrid_presentation_get_mood_name(presentation, x, y,
&name);
    gnsdk_moodgrid_presentation_find_recommendations(presentation,
provider, x, y, &results);
    gnsdk_moodgrid_results_count(results, &count);
    for ( every item i in results )
        gnsdk_moodgrid_results_enum(results, i, &ident);
}
```

Calling the `gnsdk_moodgrid_presentation_type_dimension()` function obtains the dimensions of the grid, which allows you to iterate through each cell. For each cell you can get the mood name by calling the `gnsdk_moodgrid_presentation_get_mood_name()` function with the coordinates of the cell. The locale of the mood name is based on the playlist group locale, and if that is not defined, it is based on the music group locale. You can also get the list of recommended tracks

(playlist) for that particular mood by calling the `gnsdk_moodgrid_presentation_find_recommendations()` function with the same coordinates. Finally, you can get the identifier for each track in the playlist, by iterating through the results and calling the `gnsdk_moodgrid_results_enum()` function.

### ***Filtering Mood Results***

If you wish, you can filter the Mood results. Mood provides filtering by genre, origin, and era. 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.

```
gnsdk_moodgrid_presentation_filter_set(presentation, "FILTER", GNSDK_MOODGRID_FILTER_LIST_TYPE_GENRE, GNSDK_LIST_MUSIC_GENRE_ROCK, GNSDK_MOODGRID_FILTER_CONDITION_INCLUDE);
```

### ***Shutting Down Mood***

When you are finished using Mood, you can use the `gnsdk_moodgrid_shutdown()` function to shut it down. The GNSDK modules necessary for running Mood should be shut down in the following order:

1. Mood
2. Playlist
3. MusicID
4. SQLite
5. GNSDK Manager

### ***Example: Working with Mood***

This example provides a complete Mood sample application.

Sample Application: `moodgrid/main.c`.

#### **Sample output:**

```
GNSDK Product Version : 3.05.0.798 (built 2013-05-08 16:09-0700)
```

```
Loading sample collection
Enumerating for the first Moodgrid Provider available
GNSDK_MOODGRID_PROVIDER_NAME : sample_collection

GNSDK_MOODGRID_PROVIDER_TYPE : gnsdk_playlist

GNSDK_MOODGRID_PROVIDER_NETWORK_USE : FALSE

PRINTING MOODGRID 5 x 5 GRID

  X:1  Y:1 name: Peaceful count: 10

    X:1 Y:1
ident: 20_7
group: sample_collection

    X:1 Y:1
ident: 9_7
group: sample_collection

    X:1 Y:1
ident: 11_5
group: sample_collection

    X:1 Y:1
ident: 16_11
group: sample_collection

    X:1 Y:1
ident: 18_2
group: sample_collection

    X:1 Y:1
ident: 18_8
group: sample_collection

    X:1 Y:1
ident: 19_2
group: sample_collection

    X:1 Y:1
ident: 19_8
group: sample_collection
```

```
    X:1 Y:1
ident:  20_2
group:  sample_collection

    X:1 Y:1
ident:  6_3
group:  sample_collection

    X:1  Y:2 name: Easygoing count: 3

    X:1 Y:2
ident:  18_11
group:  sample_collection

    X:1 Y:2
ident:  4_12
group:  sample_collection

    X:1 Y:2
ident:  4_4
group:  sample_collection

    X:1  Y:3 name: Upbeat count: 8

    X:1 Y:3
ident:  3_19
group:  sample_collection

    X:1 Y:3
ident:  3_2
group:  sample_collection

    X:1 Y:3
ident:  3_4
group:  sample_collection

    X:1 Y:3
ident:  3_5
group:  sample_collection

    X:1 Y:3
ident:  3_13
group:  sample_collection
```

```
    X:1 Y:3
ident:  3_14
group:  sample_collection

    X:1 Y:3
ident:  3_18
group:  sample_collection

    X:1 Y:3
ident:  3_1
group:  sample_collection

... More
```

### ***Accessing Mood and Tempo Metadata***

In GNSDK, access sonic and mood attribute metadata from Gracenote Service by setting the GNSDK\_MUSICID\_OPTION\_ENABLE\_SONIC\_DATA option to True when performing MusicID Album queries. Setting this option causes the mood and tempo metadata contained in a GDO to be automatically rendered as XML output:

An application must be entitled to implement sonic attribute metadata. Contact your Gracenote representative for more information.

## **3.12 Mood**

GNSDK provides up to two levels of granularity for mood metadata: Level 1 and Level 2.

Use the GNSDK\_GDO\_VALUE\_MOOD\_\* value keys to access available mood information from an audio track GDO , as shown below in the accessor code sample:



- **GNSDK\_GDO\_VALUE\_MOOD\_LEVEL1:** Value key to access the Level 1 mood classification, such as Blue.
- **GNSDK\_GDO\_VALUE\_MOOD\_LEVEL2:** Value key to access the Level 2 mood classification, such as Gritty/Earthy/Soulful.

For example:

```
gnsdk_manager_gdo_value_get(track_gdo, GNSDK_GDO_VALUE_MOOD_LEVEL1, 1,
&mood_level1);
gnsdk_manager_gdo_value_get(track_gdo, GNSDK_GDO_VALUE_MOOD_LEVEL2, 1,
&mood_level2);
printf("TRACK %s MOOD:\t\t\t%s (%s)\n", trknum_str, mood_level1, mood_
level2);
```

### 3.13 Tempo



**Note:** Tempo metadata is available online-only.

GNSDK provides up to three levels of granularity for tempo metadata.

Use the following **GNSDK\_GDO\_VALUE\_TEMPO\_\*** value keys to access available tempo information from an audio track GDO , as shown below in the accessor code sample:

- **GNSDK\_GDO\_VALUE\_TEMPO\_LEVEL1:** Value key to access the Level 1 tempo classification, such as Fast Tempo.
- **GNSDK\_GDO\_VALUE\_TEMPO\_LEVEL2:** Value key to access the Level 2 tempo classification, such as Very Fast.
- **GNSDK\_GDO\_VALUE\_TEMPO\_LEVEL3:** Value key to access the Level 3 tempo classification, which may be displayed as a numeric tempo range, such as 240-249, or as a descriptive phrase.

For example:

```
gnsdk_manager_gdo_value_get(track_gdo, GNSDK_GDO_VALUE_TEMPO_LEVEL1,
1, &tempo_level1);
gnsdk_manager_gdo_value_get(track_gdo, GNSDK_GDO_VALUE_TEMPO_LEVEL2,
1, &tempo_level2);
gnsdk_manager_gdo_value_get(track_gdo, GNSDK_GDO_VALUE_TEMPO_LEVEL3,
```

```
1, &tempo_level3);  
printf("TRACK %s TEMPO:\t\t\t%s (%s/%s/%s)\n", trknum_str, value,  
tempo_level1, tempo_level2, tempo_level3);
```

Here is an example list of tempo levels:

LEVEL 1	LEVEL 2	LEVEL 3
Slow Tempo		
	Very Slow	30s
	Slow	40s
Medium Tempo		
	Medium Slow	50s
	Medium	60s
		70s
		80s
	Medium Fast	90s
		100s
		110s
		120s
Fast Tempo		
	Fast	130s
		140s
		150s
		160s
		170s
	Very Fast	220s
		230s
		280s

## Playlist PDL Specification

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.

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.

### ***3.13.1 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

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

### 3.13.1.1 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>]]

```

### 3.13.2 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	+, -, *, /

### 3.13.3 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.

### 3.13.4 Attributes

Most attributes require a Gracenote-defined numeric identifier value or GDO (GnDataObject in object-oriented languages) for 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 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

### 3.13.5 Functions

PDL supports these functions:

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

### 3.13.6 PDL Statements

This topic discusses PDL statements and their components.

#### 3.13.6.1 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 ids are in the playlist results. ORDER statements that evaluate to a non-zero score determine how ids display in the playlist results.

### 3.13.6.2 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)

### 3.13.6.3 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"
```

### 3.13.6.4 Implementing Rhythm

The Rhythm API:

- Creates radio stations based on GDO seeds.
- Supports playlist creation either within radio stations, or just with GDO seeds.
- Provides for a feedback event mechanism for playlist content modification.

Implementing Rhythm in an application involves the following steps:

1. Initializing the Rhythm module.
2. Creating a Rhythm query.
3. Providing a GDO seed.
4. Creating a radio station (optional if you only want a playlist).
5. Retrieving playlist.

6. Providing feedback events to radio station.
7. Observing feedback effects by retrieving updated playlist from radio station.
8. Saving a radio station for later retrieval.

## Initializing the Rhythm Module

Before using Rhythm, follow the usual steps to initialize GNSDK. The following GNSDK modules must be initialized:

- GNSDK Manager
- MusicID (optional to retrieve GDOs)
- Playlist (optional to use generated playlists)
- Rhythm

For more information on initializing the GNSDK modules, see “Initializing an Application.”

To initialize Rhythm, use the `gnsdk_rhythm_initialize()` function.

```
error = gnsdk_rhythm_initialize(sdkmgr_handle);
```



**Note:** If you are using MusicID to recognize music, you must enable Playlist and DSP data in your query.

## Creating a Rhythm Query

Prior to generating a playlist or creating a radio station, you must create a Rhythm query. A Rhythm query requires a seed GDO, which can be created with MusicID, or provided through some other means. Once you have a seed GDO, you call `gnsdk_rhythm_query_create()` to create a query handle, and then `gnsdk_rhythm_query_set_gdo()` to set the seed GDO into the query. For example, the following code calls the appropriate functions to create a Rhythm query:

```
error = gnsdk_rhythm_query_create(user_handle, GNSDK_NULL, GNSDK_NULL,
&rhythm_query_handle);
if (!error)
    error = gnsdk_rhythm_query_set_gdo(rhythm_query_handle, seed_gdo);
```

## Creating Recommendations For Queries and Radio Station Playlists

Once you have a query with a seeded handle, you can create recommendations (in the form of a playlist) for that query or a radio station. Creating a playlist for a query uses less overhead, while creating a radio station allows for playlist changes through feedback or tuning. To generate a Rhythm query playlist, call `gnsdk_rhythm_query_generate_recommendations()` as shown below:

```
error = gnsdk_rhythm_query_generate_recommendations(rhythm_query_handle, &rhythm_playlist);
```

Once you have a playlist, you can traverse the GDOs in it to retrieve track information.

To create a radio station, call `gnsdk_rhythm_query_generate_station()` after you have set the GDO for the query. The following code shows the creation of a query, setting of a GDO, and creation of a radio station:

```
error = gnsdk_rhythm_query_create(user_handle, GNSDK_NULL, GNSDK_NULL, &rhythm_query_handle);
if (!error)
    error = gnsdk_rhythm_query_set_gdo(rhythm_query_handle, seed_gdo);
if (!error)
    error = gnsdk_rhythm_query_generate_station(rhythm_query_handle, GNSDK_NULL, GNSDK_NULL, &rhythm_station_handle);
```

The station handle is required for later use in providing feedback and tuning. To generate a radio station playlist, use the function `gnsdk_rhythm_station_generate_playlist()`. The following example shows the creation of a radio station playlist:

```
gnsdk_rhythm_station_generate_playlist(rhythm_station_handle, &rhythm_playlist);
```

## Providing Feedback

Once you have set the seed and created a station, you can not alter the seed. If you wish to change the seed for a radio station, you must create a new station.

However, you can alter an already existing station based on feedback events from the end user. The following are supported feedback types and their effect on a radio station playlist:

- **GNSDK\_RHYTHM\_EVENT\_TRACK\_PLAYED** - Track marked as played. Moves the play queue (drops track being played and adds additional track to end of queue)
- **GNSDK\_RHYTHM\_EVENT\_TRACK\_SKIPPED** - Track marked as skipped. Moves the play queue.
- **GNSDK\_RHYTHM\_EVENT\_TRACK\_LIKE** - Track marked as liked. Does not move the play queue.
- **GNSDK\_RHYTHM\_EVENT\_TRACK\_DISLIKE** - Track marked as disliked. Refreshes the playlist queue.
- **GNSDK\_RHYTHM\_EVENT\_ARTIST\_LIKE** - Artist marked as liked. Does not move the play queue.
- **GNSDK\_RHYTHM\_EVENT\_ARTIST\_DISLIKE** - Artist marked as disliked. Refreshes the playlist queue.

You provide feedback to Rhythm with the fuction `gnsdk_rhythm_station_event()`. You must provide the radio station handle, event, and specific GDO that is affected by the feedback. For example, the following code sends an artist like event back to Rhythm:

```
// retrieve album 1 GDO from playlist
error = gnsdk_manager_gdo_child_get(rhythm_playlist, GNSDK_GDO_CHILD_
ALBUM, 1, &album_gdo);
// retrieve artist GDO
if (!error)
    error = gnsdk_manager_gdo_child_get(album_gdo, GNSDK_GDO_CHILD_
ARTIST, 1, &artist_gdo);
// send feedback that we like this artist
if (!error)
    error = gnsdk_rhythm_station_event(rhythm_station_handle, GNSDK_
RHYTHM_EVENT_ARTIST_LIKE, artist_gdo);
```

## Tuning Playlists

You can adjust a radio station or query playlist by setting options for Rhythm to follow. Use the function `gnsdk_rhythm_station_option_set()` to adjust playlist options for radio stations, or `gnsdk_rhythm_query_option_set()` for queries. Options are:

- **GNSDK\_RHYTHM\_OPTION\_ENABLE\_EXTERNAL\_IDS** - Response should include external IDs, which are 3rd-party IDs to external content (for example, Amazon)
- **GNSDK\_RHYTHM\_OPTION\_ENABLE\_CONTENT\_DATA** - Response should include data for fetching content (for example, images)
- **GNSDK\_RHYTHM\_OPTION\_ENABLE\_SONIC\_DATA** - Response should include sonic attribute data. You must be licensed for this data.
- **GNSDK\_RHYTHM\_OPTION\_RETURN\_COUNT** - How many tracks to return in playlist, range is 1-25, default is 5.
- **GNSDK\_RHYTHM\_OPTION\_FOCUS\_POPULARITY** - Playlist track popularity. Range is 0-1000 (most popular). Default is 1000.
- **GNSDK\_RHYTHM\_OPTION\_FOCUS\_SIMILARITY** - Playlist track similarity. Range is 0-1000 (most similar). Default is 1000.

The following two options only affect query recommendations:

- **GNSDK\_RHYTHM\_OPTION\_RECOMMENDATION\_MAX\_TRACKS\_PER\_ARTIST** - Specifies a maximum number of tracks per artist for recommended playlist results.
- **GNSDK\_RHYTHM\_OPTION\_RECOMMENDATION\_ROTATION\_RADIO** - Enabling this option will cause results to be sequenced in a radio-like fashion, otherwise, you might, for example, get a number of tracks from the same artist (not 'radio-like'). Note that enabling this is not the same thing as creating a radio station - each recommendation query is likely to return the same or similar tracks given the same or similar seeds. Each recommendation query is considered a standalone query, and does not take into account any previous recommendation queries.

The following option only affects radio stations:

- **GNSDK\_RHYTHM\_OPTION\_STATION\_DMCA** - When creating a radio station or getting recommendations, you have the option to enable DMCA (Digital Millennium Copyright Act) rules, which reduces the repetition of songs and albums in conformance with DMCA guidelines.

For example, the following code turns DMCA support on:

```
gnsdk_rhythm_station_option_set(rhythm_station_handle, GNSDK_RHYTHM_OPTION_STATION_DMCA, "1");
```

## Saving Radio Stations

Rhythm saves all created radio stations within the Gracenote service. Stations can be recalled through the station ID, which an application can store locally. To retrieve the station ID for storage, call `gnsdk_rhythm_station_id()`.

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 handle, call `gnsdk_rhythm_station_lookup()`, and pass the station ID and user handle. The look up of the station ID can happen during the same execution or any later execution of the Rhythm software.

For example, the following code retrieves the station ID using the station handle, then uses it to retrieve the station handle:

```
// retrieve station ID
gnsdk_rhythm_station_id(rhythm_station_handle, &station_id);

// save the ID for later retrieval...

// look up station handle given station ID
gnsdk_rhythm_station_lookup(station_id, user_handle, GNSDK_NULL,
GNSDK_NULL, &rhythm_station_handle);

// use the handle for other Rhythm function...
```

## Shutting Down Rhythm

When you are finished using Rhythm, you can use the `gnsdk_Rhythm_shutdown()` function to shut it down. Shutdown the GNSDK modules you are using in the reverse order in which they were initialized.

### Sample Application

A sample application for Rhythm is provided in `rhythm/main.c`.

#### 3.13.6.5 Deploying Android Applications

To access GNSDK in an Android application, add the GNSDK jar and native shared libraries to your application's `libs` folder. Copy the jar libraries directly into the `libs` folder. Copy the native shared libraries into the architecture sub-folders under `libs`.

The table below shows the GNSDK Android libraries and their location in the package

Jar	Location
<code>gnsdk.jar</code> and <code>gnsdk_helpers.jar</code>	<code>.../wrappers/gnsdk_java/jar/android</code>
<code>libgnsdk_marshall.so</code>	<code>.../wrappers/gnsdk_java/lib/android_*</code>
<code>libgnsdk_*.so</code>	<code>.../lib/android_*</code>

GNSDK uses GABI++ for C++ support and requires that `libgabi++_shared.so` is included in the application's architecture sub-folder under `libs`. The `libgabi++_shared.so` is an Android library provided in the Android NDK

### GNSDK Android Permissions

To use the GNSDK properly in your Android application, configure the application with the follow settings:

- Record audio
- Write to external storage



- Access Internet
- Access network state

Add these permissions to the Android application's AndroidManifest.xml file.

## 3.14 Testing an Application

Gracernote helps you validate and ship your Gracernote-enabled application. Read the following sections carefully to learn the procedures you must follow to release your application.

### 3.14.1 Enabling Test Mode

When you are ready to begin testing your Gracernote-enabled application, send an email to [DevSupport@gracernote.com](mailto:DevSupport@gracernote.com) to register your client application. Gracernote support will send back information that will allow you to test your application.

### 3.14.2 Setting Environment Variables

Some GNSDK options can be set via environment variables, to allow you to enable certain features to improve your application testing. Using environment variables means that these options do not need to be accessed through your application.

GNSDK examines the environment variables listed in the following table:

Environment variable	Description
GNSDK_ASYNC_LOGGING=TRUE/FALSE	By default, logging is performed on a separate thread. If you require that no extra threads be created, or just want the logging to operate synchronously (log messages committed to file immediately), set this variable to FALSE.

Environment variable	Description
GNSDK_DEBUG_LOG=<log filename>	Set this variable to a valid file name to enable GNSDK debug logging to the given file name.
GNSDK_PROXY_HOST=hostname:port	Set this variable to set GNSDK to connect through a proxy server. If the proxy server also requires a user name and password also set those variables.
GNSDK_PROXY_USER=<proxy username>	Sets a user name for the proxy server.
GNSDK_PROXY_PASS=<proxy password>	Sets a password for the proxy server.

### 3.14.3 Enabling Full Access to Gracenote Service

When you are finished with development, ship your device to Gracenote for validation. Be sure to include any access codes or registration information needed to unlock your application for execution.

Gracenote ensures that Gracenote metadata is exchanged properly between your device and Gracenote Service. Gracenote also verifies that the device, and any packaging, is in compliance with the terms of the License Agreement. This is not a bug-testing service—it is a verification process to protect the integrity and display of Gracenote metadata.

Once Gracenote has verified your device and received your signed License Agreement, it enables the full complement of registered users (as stated in your License Agreement) for the application. For more information, refer to your license agreement.

## 3.15 Submitting New and Updated Content

Using the Submit module, an application can send new or updated Album content to Gracenote Services, including Album and Track metadata, and Track Feature data, such as audio fingerprints, and descriptors (genre, mood, tempo, and so on).

Common use cases for Submit are:

- Entering information for a new Album that is not known to Gracenote Service.
- Editing the metadata of an Album currently existing in Gracenote Service. This is equivalent to editing the information returned in an ALBUM\_RESPONSE.
- Submitting Track Feature data.

### 3.15.1 Submit Terminology

Submit uses the following terms for its components and processes:

Concept	Description
Editable GDO	A GDO with editable metadata; this is an augmentation of the current GNSDK read-only GDO model. Note that only specific fields are editable to preserve the GDO's data integrity with Gracenote Service.
Features	Term used to encompass a particular media stream's characteristics and attributes; this is data 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 data (Attributes).
Parcel	A Submit container that wraps around editable GDO and feature data to enable the data transfer to Gracenote Service. Parcels can contain a single item—only editable GDOs, or only features—or both editable GDOs and features. Submitting a parcel is analogous to performing a query in other GNSDK modules—the difference is the interaction and end result with Gracenote Service. In most other GNSDK modules, you create and define queries with specific options and inputs to access data from Gracenote Service. In Submit, you create a parcel, add data, and upload the parcel to Gracenote Service.

### 3.15.2 Submit Process




The Submit process involves these basic steps:

1. Create editable GDOs and/or features.
2. Create parcels as containers for the updated content.
3. Validate the GDO data for submission. All data submitted to Gracenote must first pass a validation test. Validation ensures that the submission meets Gracenote requirements and that key field values are valid.

4. Submit parcels to Gracenote Services. The Submit module performs error checking during the submission process.
5. Provide a visual confirmation and submission summary to indicate the submit process was successful.

### **3.15.3** *Submit APIs*

Submit APIs are grouped into four functional categories: General, Album and Track Metadata, Track Features, and Parcel.

Function	Category	Usage	See Also
gnsdk_submit_*( gnsdk_submit_ get_*)	General	Perform standard tasks	
gnsdk_submit_ edit_gdo_*( )	Album and Track Metadata	Create editable GDOs, and add and maintain children, values, and list value data.	Submit, Album Data  Submit Requirements
gnsdk_submit_ parcel_ feature_*	()Track Features	Initialize, set and get options for, and process features of an audio stream.	Submit, Feature Data  Submit Requirements
gnsdk_submit_ parcel_ data_*( )	Parcels	Perform administrative tasks for parcel submits, such as creating parcels, adding information, uploading parcels, and accessing status and state information.  This group also intrinsically contains the following standard GNSDK module APIs:  <ul style="list-style-type: none"> <li> gnsdk_submit_parcel_create()</li> <li> gnsdk_submit_parcel_upload()</li> <li> gnsdk_submit_parcel_release()</li> </ul>	Submit, Parcel Submit Requirements

### 3.15.4 Editable GDO Child and Value Keys

The following table lists child and value keys for the GDO types supported by editable GDOs. The child keys are relevant for use in any gnsdk\_submit\_edit\_gdo\_child\_\*(  
) API. The value keys can be used for any gnsdk\_submit\_edit\_gdo\_value\_\*(  
) API, except for the Genre and Role list-based keys. See Editable GDO Considerations.

Context	Child Keys	Value Keys
GNSDK_GDO_CONTEXT_ALBUM	GNSDK_GDO_CHILD_CREDIT GNSDK_GDO_CHILD_TRACK	GNSDK_GDO_VALUE_ALBUM_LABEL GNSDK_GDO_VALUE_HAS_TRACK_ARTISTS GNSDK_GDO_VALUE_ALBUM_DISC_IN_SET GNSDK_GDO_VALUE_ALBUM_TOTAL_IN_SET GNSDK_GDO_VALUE_ARTIST_DISPLAY (Required) GNSDK_GDO_VALUE_ARTIST_FAMILY (Optional) GNSDK_GDO_VALUE_ARTIST_GIVEN (Optional) GNSDK_GDO_VALUE_ARTIST_PREFIX (Optional) GNSDK_GDO_VALUE_DATE GNSDK_GDO_VALUE_DATE_RELEASE GNSDK_GDO_VALUE_GENRE (Required) GNSDK_GDO_VALUE_GENRE_META (Required; setting GNSDK_GDO_VALUE_GENRE automatically sets this value.) GNSDK_GDO_VALUE_GENRE_MICRO (Required; setting GNSDK_GDO_VALUE_GENRE automatically sets this value.) GNSDK_GDO_VALUE_PACKAGE_LANGUAGE GNSDK_GDO_VALUE_PACKAGE_LANGUAGE_DISPLAY GNSDK_GDO_VALUE_TITLE_DISPLAY (Required) GNSDK_GDO_VALUE_TITLE_SORTABLE GNSDK_GDO_VALUE_TOC_ALBUM (Required)

Context	Child Keys	Value Keys
GNSDK_GDO_CONTEXT_TRACK	GNSDK_GDO_CHILD_CREDIT	GNSDK_GDO_VALUE_ARTIST_DISPLAY (Required) GNSDK_GDO_VALUE_ARTIST_FAMILY (Optional) GNSDK_GDO_VALUE_ARTIST_GIVEN (Optional) GNSDK_GDO_VALUE_ARTIST_PREFIX (Optional) GNSDK_GDO_VALUE_DATE GNSDK_GDO_VALUE_DATE_RELEASE GNSDK_GDO_VALUE_GENRE (Required) GNSDK_GDO_VALUE_GENRE_META (Required; setting GNSDK_GDO_VALUE_GENRE automatically sets this value.) GNSDK_GDO_VALUE_GENRE_MICRO (Required; setting GNSDK_GDO_VALUE_GENRE automatically sets this value.) GNSDK_GDO_VALUE_TITLE_DISPLAY (Required) GNSDK_GDO_VALUE_TITLE_SORTABLE GNSDK_GDO_VALUE_TRACK_NUMBER (Required)
GNSDK_GDO_CONTEXT_CREDIT	N/A	GNSDK_GDO_VALUE_NAME_DISPLAY (Required) GNSDK_GDO_VALUE_NAME_FAMILY (Optional) GNSDK_GDO_VALUE_NAME_GIVEN (Optional) GNSDK_GDO_VALUE_NAME_PREFIX (Optional) GNSDK_GDO_VALUE_ROLE (Required) GNSDK_GDO_VALUE_ROLE_CATEGORY (Required; setting GNSDK_GDO_VALUE_ROLE automatically sets this value.)

### 3.15.4.1 Editable GDO Considerations

When creating editable GDOs, consider these guidelines:

- The `GNSDK_GDO_VALUE_GENRE` and `GNSDK_GDO_VALUE_ROLE` keys are list-based values. List-based values are returned by Gracenote Services as numbered IDs. These numbered IDs map to string display values based on locally-available lists. GNSDK applications typically display genre and role with the value returned by their GDO key. However, to edit these values, the application must use the list-based edit API. This approach differs from non-list-based values, such as artist name or track title. Non-list-based values are displayed and edited using their GDO keys.
- To edit list-based values, use the `gnsdk_submit_edit_gdo_list_value_set_by_submit_id()` API with the appropriate list type and Submit ID. The appropriate lists types are `GNSDK_LIST_TYPE_GENRES` to edit genre values, and `GNSDK_LIST_TYPE_ROLES` to edit roles. Attempting to edit a list-based key with the GDO key-based API `gnsdk_submit_edit_gdo_value_*`() returns a `SUBMITERR_Unsupported` error.
- Specifying an ID for the appropriate list type, `GNSDK_LIST_TYPE_GENRES` or `GNSDK_LIST_TYPE_ROLES`, automatically populates all of the display-level values. For example, all Genre levels are updated with a single call to the list-based edit API.
- Keys that are supported in the parent type but cannot be changed in an editable GDO are read-only. Attempting to edit these read-only keys results in a `SUBMITWARN_NotEditable` error.
- Keys that are not supported for the given parent type (for example, a video-specific key in an Album type) result in a `SUBMITERR_Unsupported` error.

### **3.15.5 Submitting Album Metadata**

You can submit Album metadata using two methods:

- Use the `gnsdk_submit_edit_gdo_*`() functions (Recommended).
- Directly edit XML created from an XML-rendered Full album GDO, using the `gnsdk_manager_gdo_render_to_xml()` function with the `GNSDK_GDO_RENDER_XML_SUBMIT` render flag.

The Gracenote XML Parser is optimized for high efficiency, and consequently does not support certain advanced XML features. In addition to requiring correctly



escaped data, the parser does not support the following constructs: A colon (:) in attribute names, any comments: <!-- comments -->, and single quotes enclosing attribute values, such as value='1' instead of value="1"

The following table shows the Submit functions used to create editable GDOs.

Album Metadata Submit Function Family	Usage
gnsdk_submit_ edit_gdo_create_* ()	Create an editable GDO for an Album from an empty GDO, or pre-populated with information from a source GDO or from GDO -formatted XML.
gnsdk_submit_ edit_gdo_child_*)	Add child GDOs for the individual tracks to the previously-created editable GDO.
gnsdk_submit_ edit_gdo_value_*)	Add non-list-based values at the Album and Track level.
gnsdk_submit_ edit_gdo_list_ value_*)	Edit the metadata for four list-based fields (album language, album genre(s), track genre(s), and credit role(s)) by accessing and setting a list element's Submit identification number.

During the upload process, the Submit functionality validates the following information before finalizing the upload to the Gracenote Service. The first five items are required fields. Credits are optional; however, when included, each credit must contain a name and a role.

- Album TOC (Required)
- Album artist (Required)
- Album title (Required)
- Album genre (Required)
- Correct number of Track children, including Track title and Track number (Required)
- Credit, including name and role (Optional)

### 3.15.5.1 Example: Editing and Submitting an Existing Album GDO

This simplified example demonstrates editing and submitting an existing album GDO and submitting it to Gracenote Service.

Sample Code:

submit\_album/main.c

Steps for this sample application:

1. Initialize Manager, User, MusicID, Submit, and DSP.
2. Create empty GDO.
3. Set the TOC in empty GDO created (a required field).
4. Add album metadata.
5. Add album credit.
6. Add album genre.
7. Add 1st track and its metadata.
8. Display created album GDO.
9. Submit album GDO.
10. Shutdown GNSDK.

### 3.15.6 Submitting Track Features

You can submit Track Features only for Album GDOs generated from a TOC lookup. The following table shows the Submit functions used to create and process Track Features.

Function	Usage
<code>gnsdk_submit_parcel_data_init_features()</code>	Enables Gracenote Service to determine what specific feature information is needed for a particular Track.  Important: Be sure to call this function only once per Album.

Function	Usage
gnsdk_submit_parcel_feature_option_set() gnsdk_submit_parcel_feature_option_get()	Set and access options for the audio stream, such as source name and bit rate.
gnsdk_submit_parcel_feature_init_audio()	Initialize the audio stream write process. Note that you must initialize the GNSDK DSP module before calling this function.
gnsdk_submit_parcel_feature_write_audio_data()	Process the audio data; this essentially compiles the Track information to a pre-upload ready state.
gnsdk_submit_parcel_feature_finalize()	Finalize the audio stream write process.

### 3.15.6.1 Example: Submitting Track Features (1)

The example illustrates processing and submitting feature data.

Sample Application: submit\_feature/main.c

Steps for this code sample:

1. Initialize features to determine which data is required by the Gracenote Service, and must be uploaded.
2. Define each stream's audio source information.
3. Process a stream by initializing, writing, and finalizing the feature data, prior to performing the submit upload.

Be sure to call gnsdk\_submit\_parcel\_data\_init\_features() only once per Album.

### 3.15.6.2 Example: Submitting Track Features (2)

Sample Code:

submit\_feature/main.c

Steps for this sample application:

1. Initialize Manager, User, MusicID, Submit, and DSP.
2. Create MusicID album query.
3. Get album GDO.
4. Create submit parcel.
5. Initialize the features for the Album using its GDO.
6. See if there are any features to generate.
7. For all tracks of the album:
8. Set audio source, id, description, bitrate, bitrate type.
9. Read audio for track and feed it to `gnsdk_submit_parcel_feature_write_audio_data()`.
10. Finalize features.
11. Upload parcel.
12. Display upload state.

### **3.15.7** *Submitting Parcels*

The Submit APIs for parcels support the following use cases:

- Add 1-n editable GDOs or 1-n Features, or both to a parcel.
- Add a unique identifier for a GDO or a Feature used later to access callback status information.
- Bundle editable GDOs and Features together in a parcel, or upload either item separately.
- Submit a parcel numerous times.

The following table shows the Submit functions used to create and process parcels.

Function	Usage
<code>gnsdk_submit_parcel_create()</code>	Create an empty parcel and add editable GDOs, feature data, or both.

Function	Usage
<code>gnsdk_submit_parcel_data_add_gdo()</code>	Add completed editable GDOs to a parcel for Submit uploading.
<code>gnsdk_submit_parcel_upload()</code>	Upload a parcel containing editable GDOs, feature data, or both to Gracenote Service.
<code>gnsdk_submit_parcel_data_get_state()</code>	Set callback functionality to access a parcel's upload progress and network status information.

#### 3.15.7.1 Example: Submitting an Album Parcel

Sample Application: `submit_album/main.c`

Steps for this application:

1. Create an editable GDO
2. Edit the album metadata.
3. Add this GDO to a submit parcel.
4. Upload the parcel to submit the data.
5. Verify the upload status of the parcel and the GDO.

#### 3.15.8 Availability of Edited Data Cache

Edited Submit data resides in the application's lookup cache. You can implement a cache using the SQLite module, or alternately, implement your own application-specific cache.) Note that SQLite stores only the edited data, and not the entire parcel.

Submit data edits exist in the lookup cache according to the specified default expiration for Album query lookups. Generally, the default cache expiration time is three to seven (3-7) days.

Once Submit data is successfully uploaded, it takes approximately seven (7) days for Gracenote Service to make the edited data available for access.

Cached Submit edits are affected by the preferred language option active at the time the edits are performed. For example, if you cache edits to an Album that

was queried when the preferred language set was to Chinese, and then change the preferred language option to English, the edits are not visible.

### **3.15.9** *Synchronizing Dependent Fields*

Some metadata fields are dependent on other fields. Therefore editing a dependent field's metadata without also editing the fields on which it depends may cause a synchronization problem.

To avoid this problem, the Submit functionality ensures that when a field's metadata is changed, all of its dependent field metadata are also changed. If not, Submit sets the dependent field metadata to NULL during the upload process to ensure data integrity.

Some dependent fields are:

- GNSDK\_GDO\_VALUE\_TITLE\_SORTABLE must stay synchronized with GNSDK\_GDO\_VALUE\_TITLE\_DISPLAY in Album and Track types.
- GNSDK\_GDO\_VALUE\_ARTIST\_GIVEN, GNSDK\_GDO\_VALUE\_ARTIST\_PREFIX, GNSDK\_GDO\_VALUE\_ARTIST\_FAMILY must stay synchronized with GNSDK\_GDO\_VALUE\_ARTIST\_DISPLAY, in Album and Track types.
- And, GNSDK\_GDO\_VALUE\_ARTIST\_GIVEN, GNSDK\_GDO\_VALUE\_ARTIST\_PREFIX, GNSDK\_GDO\_VALUE\_ARTIST\_FAMILY must stay synchronized with GNSDK\_GDO\_VALUE\_NAME\_GIVEN, GNSDK\_GDO\_VALUE\_NAME\_PREFIX, and GNSDK\_GDO\_VALUE\_NAME\_FAMILY, in Credit types.

### **3.15.10** *Enabling Test Mode for Submit Finalization*

You must test the application's submission logic to prevent the upload of invalid data to Gracenote Service.

Set the GNSDK\_SUBMIT\_PARCEL\_UPLOAD\_FLAG\_TEST\_MODE flag to indicate the application is uploading test parcels. Once Gracenote validates the application's submission logic, clear the flag to enable actual parcel uploading.

If your application receives an error or aborts while calling the finalization function `gnsdk_submit_parcel_feature_finalize()`, be sure the application calls the upload function `gnsdk_submit_parcel_upload()`. This ensures sending important information to Gracenote that is useful for error resolution.





### **3.0.1** *API Reference Overview*

The following API Reference documentation is available for GNSDK.

API	Location in Package
C API Reference	docs/hml-docs/html-c/Content/api_ref_c/index.html



**Note:** For a list of third-party and open-source (OSS) licenses used by GNSDK, see the /docs/license-files folder in the software package.

## Chapter 4 Data Models

This section describes the data elements, attributes, and values used in GNSDK applications.

### 4.1 C Data Model

The GNSDKC data model represents Gracenote media elements and metadata. The model establishes interrelationships among Gracenote Data Objects (GDOs) and the metadata they contain or reference. A table-based version of the data model is here: [C Data Model](#).

This table maps GDO types to their corresponding Child GDOs and values. The table also links directly to the corresponding GDO and value APIs documented in the GNSDKAPI Reference. The data model table provides the following information:

Table Column	Description
GDO Elements	This column lists GDO elements for each type. Each entry links to its documentation in the API Reference.
Child and Value Keys	This column lists CHILD and VALUE keys for the GDO Element. CHILD keys return GDO types. For example, GNSDK_GDO_CHILD_ALBUM returns the type GNSDK_GDO_TYPE_ALBUM. These relationships are shown as superscripted table_ref links. Click the link to go to the associated GDO type within the table that the CHILD key returns.
Dependent List	This column lists metadata that is locale- or list-dependent. These are values that may vary depending on the locale used by query as well as general metadata that is represented as lists in the API. Examples include genre, artist origin, artist era, artist type, mood, tempo, contributor lists, roles, and others. For more information about locale-dependent values and their corresponding List types, see <i>"Using Locales" on page 106</i> .

Table Column	Description
In Partial	This column lists metadata returned in GDO partial results. For information about GDO partial and full results, see <i>"About Gracenote Data Objects (GDOs)" on page 138</i> .
Multi	This column lists elements that can contain multiple results.
Serialized	This column lists metadata returned in serialized GDOs.



## Chapter 5 API Reference

This section provides reference information about GNSDK APIs. For details about how to use these APIs, see [Develop and Implement](#).



**Note:** For a list of third-party and open-source (OSS) licenses used by GNSDK, open the /docs/license-files folder in the software package.

### 5.1 Playlist PDL Specification

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.

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.

#### 5.1.1 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.

### 5.1.1.1 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

Keyword	Description	Required or Optional	PDL Statement Order
LIMIT	<p>Specifies criteria used to restrict the number of returned results.</p> <p>Also uses the keywords RESULT and PER.</p> <p>Example: Limiting the number of tracks displayed in a playlist to 30 results with a maximum of two tracks per artist.</p> <p>The expression format is: &lt;identifier&gt; &lt;operator&gt; &lt;identifier&gt;</p>	Optional	4
SEED	<p>Specifies input data criteria from one or more ids.</p> <p>Typically, a Seed is the This in a More Like This playlist request.</p> <p>Example: Using a Seed of Norah Jones' track Don't Know Why to generate a playlist of female artists of a similar genre.</p> <p>The expression format is: &lt;identifier&gt; &lt;operator&gt; &lt;identifier&gt;</p>	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>]]

```

### 5.1.1.2 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	+, -, *, /

### 5.1.1.3 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.

### 5.1.1.4 Attributes

Most attributes require a Gracenote-defined numeric identifier value or GDO (GnDataObject in object-oriented languages) for 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 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

#### 5.1.1.5 Functions

PDL supports these functions:

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

#### 5.1.1.6 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 ids are in the playlist results. ORDER statements that evaluate to a non-zero score determine how ids 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.

## 5.1.2 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"
```

## Chapter 5 Using Docs and Resources

You can navigate to Help topics using the navigation menu at the top of each page. You can also navigate to more detailed list of topics using right-side menu and topic TOCs. Use the breadcrumb links to jump to other sections in your navigation path.

### 5.2 Searching


Use the search box to locate all topics containing the search text.

You can use partial word searches, multiple word searches, and search by phrase (put the phrase within quotes). Below are general rules for using search:

- **Full-Text Search:** Search is not case-sensitive. Searching for the word "fetch" will find matches for "Fetch" and "fetch." Search for the word "fetch" will also find matches for "fetched," "fetching," and "fetches". Since matches are not case-sensitive, the results will include topics containing matches such as, "Fetched," "Fetching," and "Fetches".
- **Phrase Search:** You can search for phrases by enclosing search terms in quotation marks.
- **Boolean Search:** You can use boolean operators in searches. Supported operators are AND, OR, NEAR, NOT, and ( ). Use operators in combination with search terms to increase or decrease the number of search results.
- **Partial-Word Search:** Similar to using a simple text search, except you can search for partial-word and number strings.

## 5.3 Using the Toolbar





Each topic page has a toolbar: 

From left to right, this toolbar contains:

- **Next:** Navigate to next topics in the current section.
- **Back:** Navigate to previous topics in the current section.
- **Search highlight:** Removes text highlights after a search.
- **Expand sections:** Expands all collapsible sections on the page.
- **Print:** Prints the current page.

## 5.4 Components and Resources

In general, Gracenote product documentation includes the following components.

Document	Description
 Help System	This Help System provides the most comprehensive product information, including API and Data Model /Schema reference content.
 Developer Guide	A PDF version of the Help. The API and schema reference content is usually excluded from the PDF Dev Guide due to its size and complexity.
 Release Notes	A PDF summary of what is new and changed in the current release.
 Other	Optionally, other PDFs, such as overviews and related information.

Each documentation set is broadly organized into general groups based on information type.

Category	Description
Concepts	General overview information about the product.
Setup/Samples	Information about setting up development environments, and using sample and reference applications.
Develop/Implement	Detailed information and code snippets to perform specific development tasks and implement Gracenote applications.
API Reference	Reference information describing the product APIs.
Data Model Schema Reference	Reference information about Gracenote data elements and attributes, and their interrelationships. May also include controlled vocabulary lists.
Source Files	Where needed, the documentation includes source files, such as XML schema xsds and/or json schemas



## Glossary

### A

---

#### AV Work

In general, the terms Audio-Visual Work, Work, and AV Work are interchangeable. A Work refers to the artistic creation and production of a Film, TV Series, or other form of video content. The same Work can be released on multiple Product formats across territories. For example, The Dark Knight Work can be released on a Blu-ray Product in multiple countries. A TV Series such as Lost is also a Work. Each individual episode comprising the Series is also a unique Work. Although the majority of Works are commercially released as a Product, not all Works have a Product counterpart. For example, a TV episode which airs on TV, but is not released on DVD or Blu-ray is considered a Work to which no Product exists.

### C

---

#### Chapter

A Video feature may contain chapters for easy navigation and as bookmarks for partial viewing.

#### Character

An imaginary person represented in a Video Work of fiction. Feature - A video feature has a full-length running time usually between 60 and 120 minutes. A feature is the main component of a DVD or Blu-ray disc which may, in addition, contain extra, or bonus, video clips and features.

## Contributor

A Contributor refers to any person who plays a role in a 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.

## Credit

A credit lists the contribution of a person (or occasionally a company, such as a film studio) to a Video Work.

## E

---

## Episode

A specific instance of a TV Program in a TV Series.

## F

---

## Filmography

All of the Works associated with a Contributor in the Gracenote Service, for example: All Works that are linked to Tom Hanks. Franchise - A collection of related Video Works. For example: Batman, Friends, Star Wars, and CSI.

## Filmography

All of the Works associated with a Contributor in the Gracenote Service, for example: All Works that are linked to Tom Hanks.

## Franchise

A collection of related Works. Each of the following examples is a unique franchise: Batman Friends Star Wars CSI



## M

---

### Mediography

All of the Works associated with a Contributor in the Gracenote Service, for example: All works that are linked to Tom Hanks.

## P

---

### Product

A Product refers to the commercial release of a Film, TV Series, or video content. Products contain a unique commercial code such as a UPC, Hinban, or EAN. Products are for the most part released on a physical format, such as a DVD or Blu-ray.

## S

---

### Season

An ordered collection of Works, typically representing a season of a TV series. For example: CSI: Miami (Season One), CSI: Miami (Season Two), CSI: Miami (Season Three)

### Series

A collection of related Works, typically in sequence, and often comprised of Seasons (generally for television series); for example, CSI Las Vegas. A Series object may have varying structures of Episodes and Seasons objects. Three common Series object structure hierarchies are - Series object contains only Episode objects; for example, Ken Burns' The Civil War series. Series object contains multiple Seasons objects that contain multiple Episodes objects; for example, the animated television series The Simpsons. Series object contains one or more independent Episodes objects (meaning, not contained within Seasons objects) and multiple

Seasons objects that contain multiple Episodes objects. This structure occurs for cases when a pilot episode (represented by an independent Episode object) is developed into a series. An example of this is Cartoon Network's Samurai Jack series, which was initially launched as a television movie and then later developed into a television series.

---

## T

### Title

Also referred to as Title Set. DVDs and Blu-ray discs may contain multiple titles. A typical movie DVD may be comprised of multiple titles, one of which comprises the main feature (in this case, the movie) and is referred to as the main title. Other titles, or extras, are often comprised of previews, behind-the-scenes documentaries, or other content.

### TOP

Table of Programs. The list of program addresses found on a DVD that allows the GN MusicID system to find the DVD in the MDB.

---

## V

### Video Clip

A short video presentation.

### Video Disc

A video disc can be either DVD (Digital Video Disc) or Blu-ray. DVD is an optical disc storage format, invented and developed by Philips, Sony, Toshiba, and Panasonic in 1995. DVDs offer higher storage capacity than Compact Discs while having the same dimensions. Blu-ray is an optical disc format designed to display high definition video and store large amounts of data. The name Blu-ray Disc refers to the blue laser used to

read the disc, which allows information to be stored at a greater density than is possible with the longer-wavelength red laser used for DVDs.

### Video Explore

Provides extended video recognition and searching, enabling user exploration and discovery features.

### Video ID Module

Provides video item recognition for DVD and Blu-ray products via TOCs.

### Video Layer

Both DVDs and Blu-ray Discs can be dual layer. These discs are only writable on one side of the disc, but contain two layers on that single side for writing data to. Dual-Layer recordable DVDs come in two formats: DVD-R DL and DVD+R DL. They can hold up to 8.5GB on the two layers. Dual-layer Blu-ray discs can store 50 GB of data (25GB on each layer).

### Video Side

Both DVDs and Blu-ray discs can be dual side. Double-Sided discs include a single layer on each side of the disc that data can be recorded to. Double-Sided recordable DVDs come in two formats: DVD-R and DVD+R, including the rewritable DVD-RW and DVD+RW. These discs can hold about 8.75GB of data if you burn to both sides. Dual-side Blu-ray discs can store 50 GB of data (25GB on each side).

## W

---

### Work

See AV Work



## Index

### A

ACR [45](#)

Album [6](#), [10](#), [13](#), [19](#), [23](#), [43](#), [57](#), [74](#), [83](#), [139](#), [145](#), [149](#), [155](#), [160](#), [163](#), [166](#), [169](#), [189](#), [212](#), [215](#), [224-225](#), [231](#), [255](#), [263](#), [273-275](#), [279](#), [281-285](#), [296](#)

AlbumID [15](#), [18](#), [49](#), [67](#), [179](#), [191](#)

Android [53](#), [271](#)

API Reference [105](#), [133](#), [141](#), [144](#), [154](#), [288-289](#), [292](#), [300](#)

Artist [2](#), [6](#), [19](#), [38](#), [41](#), [125](#), [139](#), [156](#), [160](#), [173](#), [184](#), [224-225](#), [263](#), [268](#), [296](#)

    Name [8](#), [160](#), [173](#), [263](#), [296](#)

Audio [11](#), [199-200](#)

Authenticate [155](#), [167](#), [169](#), [178](#), [189](#), [198](#), [207](#)

AV Work [2](#), [42](#)

Availability [284](#)

### B

Batch [174](#), [202](#)

Best Practices [13](#), [16](#), [42](#), [116](#), [172](#), [200](#), [207](#), [242](#)

### C

C API [105](#), [141](#), [144](#), [154](#), [288](#)

Cache [284](#)

Callback [102](#), [180](#), [200](#)

Catalog [7](#)  
CD TOC [11](#), [17](#), [46](#), [138](#), [166](#), [175](#), [203](#), [222](#), [224](#), [226](#), [237](#)  
Child GDO [85](#)  
Child Key [146](#)  
Classical Music [6](#), [212](#)  
Client ID [54](#), [78](#), [89](#)  
Client Tag [78](#)  
Collaborative Artists [207](#), [218](#)  
Collection [20](#), [22](#), [24](#), [208](#), [235](#)  
Contributor [42](#), [76](#), [151](#), [156](#), [169](#), [173](#), [208](#)  
Correlates [118](#)  
Cover Art [40](#), [43](#), [225-226](#)  
Credit [76](#), [156](#), [285](#)

## D

Data Dictionary [84](#)  
Data Model [141](#), [144](#), [154](#), [212](#), [231](#), [289](#), [300](#)  
Database [141](#)  
Dependent Fields [285](#)  
Descriptor [107](#)  
Dimensions [39](#), [227](#)  
Discovery [125](#)  
DMCA [270](#)  
DSP [14](#), [46-47](#), [55](#), [90](#), [179](#), [189](#), [198](#), [200](#), [250](#), [266](#), [281-283](#)

## E

Enriched Content [223](#), [225](#), [230](#)

Enriched Metadata [5](#)

Enrichment [11](#), [19](#)

Environment [53](#), [70](#), [272](#)

Environment Variables [272](#)

EPG [45](#), [107](#)

Equivalency [22](#)

Era [6-7](#), [22](#), [76](#), [125](#), [156](#), [212](#), [244](#), [263](#), [296](#)

Event [102](#)

External IDs (XIDs) [215](#)

## F

Fetch [223](#), [225](#), [299](#)

FileInfo [179](#), [189](#)

Find [57](#), [84](#), [186](#)

Fingerprint [13](#), [189](#)

## G

GDB [108](#), [132](#)

GDO [17](#), [23](#), [50](#), [68](#), [87](#), [89](#), [109](#), [119](#), [128](#), [143](#), [149](#), [155](#), [159](#), [163](#), [168](#), [181](#), [188](#), [212](#), [215](#), [224](#), [226](#), [230-231](#), [255](#), [262](#), [265](#), [274](#), [276](#), [278-279](#), [281](#), [283-285](#), [295](#)

Genre [2](#), [10](#), [22](#), [31](#), [38](#), [41](#), [76](#), [110](#), [121](#), [125](#), [156](#), [224-225](#), [244](#), [263](#), [276](#),  
[279](#), [296](#)

GnDataObject [262](#), [295](#)

GNSDK\_GDO\_VALUE\_COLLABORATOR\_RESULT [220](#)

GNSDK\_LOOKUP\_MODE\_LOCAL [124](#), [127](#), [182](#), [224](#)

GNSDK\_LOOKUP\_MODE\_ONLINE [115](#), [123](#), [177](#), [206](#), [224](#)

GNSDK\_LOOKUP\_MODE\_ONLINE\_CACHEONLY [124](#)

GNSDK\_LOOKUP\_MODE\_ONLINE\_NOCACHE [177](#), [206](#), [224](#)

GNSDK\_LOOKUP\_MODE\_ONLINE\_NOCACHEREAD [123](#)

GOET [262](#), [296](#)

Gracenote [ii](#), [1-2](#), [5-6](#), [9-11](#), [13](#), [17](#), [19-20](#), [23](#), [25](#), [28](#), [37](#), [39](#), [42](#), [45](#), [48](#), [52](#),  
[65](#), [71](#), [74](#), [78](#), [82](#), [89](#), [91](#), [95](#), [97](#), [100](#), [106](#), [117](#), [123](#), [126](#), [128](#), [130](#), [133](#),  
[137-138](#), [143](#), [155](#), [159](#), [164](#), [166](#), [174](#), [200](#), [202](#), [208](#), [212](#), [215](#), [221-222](#),  
[228](#), [230](#), [237](#), [249](#), [255](#), [261](#), [270](#), [272-274](#), [279](#), [281-282](#), [284-285](#), [289](#),  
[294](#), [299](#)

Gracenote Data Object (GDO) [89](#)

Gracenote Media Elements [2](#)

Groups [5](#), [109](#)

## H

Header Files [90](#)

Hierarchical Groups [5](#)

## I

Identifiers [9](#), [23](#), [159](#), [165](#), [236](#)

Comparison Matrix [161](#)



Identify [18](#)

Image

Dimensions [40](#)

Formats [39](#), [227](#)

Images [40](#), [226](#)

Initialize [56](#), [78](#), [89](#), [167](#), [169](#), [179](#), [189](#), [215](#), [226](#), [243](#), [276](#), [281-283](#)

Introduction [52](#)

iOS [53](#)

Iterate [186](#), [242](#)

## L

LANG [82](#), [107](#), [212](#), [231](#)

Language [20](#), [46](#), [76](#), [107](#), [118](#), [156](#), [235](#), [258](#), [292](#)

Library Files [53](#)

Library Manager [80](#)

LibraryID [15](#), [50](#), [67](#), [179](#), [194](#)

License File [78](#), [96](#)

Link [9](#), [19](#), [45](#), [50](#), [68](#), [90](#), [125](#), [138](#), [223](#), [225](#)

Linux [53](#), [73](#)

List [2](#), [8](#), [104](#), [237](#), [279](#), [289](#)

Literals [262](#), [295](#)

Load Balancing [137](#)

Local [45](#), [79](#), [90](#), [122](#), [124](#), [126](#), [128](#), [134](#), [141](#), [166](#), [227](#), [238](#)

Local Database [141](#)

Local Lookup [79](#), [127](#), [166](#)

Local Storage [122](#), [238](#)

Locale [104](#), [118](#), [169](#), [243](#)

Locale-Dependent [110](#)

Localization [82](#)

Log [71](#)

Lookup [145](#)

## M

MacOS [74](#)

Manager [45](#), [49](#), [56](#), [66](#), [74](#), [78](#), [89-90](#), [95](#), [102](#), [116](#), [118](#), [134](#), [156](#), [169](#), [178](#),  
[180](#), [189](#), [198](#), [207](#), [223](#), [225](#), [250](#), [262](#), [266](#), [281](#), [283](#), [295](#)

Match [13](#), [39](#), [76](#), [90](#), [139](#), [143](#), [153](#), [155](#), [167](#), [170](#), [172](#), [189](#), [198](#)

Score [142](#)

Media [2](#), [10](#), [23](#), [40](#), [44](#), [91](#), [182](#), [243](#)

Media File [182](#)

Memory [96](#)

Metadata [5-6](#), [14](#), [23](#), [39](#), [86](#), [117](#), [125](#), [140](#), [156](#), [182](#), [212](#), [218](#), [231](#), [255](#), [275](#),  
[279](#)

Microphone [199](#)

Module [223](#), [225](#), [249](#), [266](#)

Mood [6](#), [10](#), [22-23](#), [25](#), [27](#), [31](#), [49](#), [67](#), [90](#), [104](#), [118](#), [125](#), [244](#), [249](#), [255](#), [263](#),  
[296](#)

MoodGrid [46](#), [51](#), [69](#)

Multi-Threaded Access [114](#)

Multiple TOC Matches [12](#)

Multiple TOCs and Fuzzy Matching [12](#)

Music [1-2](#), [6](#), [9](#), [13](#), [19](#), [40](#), [46](#), [106](#), [121](#), [160](#), [165-167](#), [174](#), [188](#), [196](#), [202](#),  
[210](#), [212](#), [224](#), [226](#), [231](#)

Music GDOs [210](#)

MusicID [10-11](#), [13-14](#), [17-18](#), [21](#), [23](#), [25](#), [38-39](#), [46](#), [49](#), [55](#), [67](#), [74](#), [78-79](#), [82](#),  
[86](#), [89-91](#), [104](#), [125-126](#), [138](#), [155](#), [166](#), [169](#), [172](#), [174](#), [179](#), [188](#), [197](#),  
[199-200](#), [203](#), [212](#), [231](#), [235](#), [250](#), [255](#), [266](#), [281](#), [283](#)

MusicID-File [14](#), [17](#), [21](#), [23](#), [39](#), [46](#), [49](#), [55](#), [67](#), [125](#), [179](#), [188](#), [212](#), [235](#)

## N

Name [49](#), [67](#), [250](#)

Navigate and Parse Additional Child GDOs [86](#)

Navigating [27](#), [76](#), [156](#), [210](#), [218](#)

Navigation and Display [83](#)

Navigation Examples [155](#)

Network [97](#), [251](#)

Number [22](#), [76](#), [156](#), [183](#)

## O

Order [229](#), [260](#), [293](#)

Origin [76](#), [125](#), [156](#), [244](#), [263](#), [296](#)

Overview [10](#), [14](#), [25](#), [37](#), [39](#), [45](#), [179](#), [231](#), [288](#)

## P

Parcel [274-275](#), [284](#)

Parse [86](#), [187](#)

Partial [52](#), [84](#), [140](#), [150](#), [290](#), [299](#)

PDL [24](#), [258](#), [292](#)

Performance [242](#)

Permissions [271](#)

Physical Media [243](#)

Platform [70](#)

Platform-Specific [71](#)

Play [37](#)

Playlist [6](#), [20](#), [22](#), [30](#), [38](#), [46](#), [49](#), [67](#), [73](#), [90-91](#), [114](#), [118](#), [125](#), [234](#), [250](#), [258](#), [266](#), [292](#)

Populate [234](#)

Process [89](#), [182](#), [242](#), [274](#), [282](#)

Product [2](#), [12](#), [42](#), [142](#), [155](#), [167](#), [170](#), [189](#), [198](#), [212](#), [244](#), [252](#)

Providers [122](#), [249](#)

Proxy [97](#)

Proxy Server [97](#)

## Q

Query [39](#), [57](#), [76](#), [84](#), [87](#), [130](#), [152](#), [155](#), [167](#), [174](#), [180](#), [198](#), [203](#), [212](#), [231](#), [258](#), [266](#), [292](#)

## R

Radio [37](#), [128](#), [199](#), [212](#), [267](#)

Region [107](#), [118](#)

Registration [81](#)

Requirements [20](#), [70](#), [276](#)

Response [15](#), [84](#), [139](#), [145](#), [181](#), [194](#), [269](#)

Response GDO [84](#), [146](#), [181](#)

Results [31](#), [129](#), [138](#), [152](#), [168](#), [179](#), [219](#), [240](#), [252](#)

Rhythm [35](#), [37](#), [47](#), [51](#), [68](#), [90](#), [265](#)

Role [276](#)

## S

Sample Application [69](#), [74](#), [97](#), [106](#), [117](#), [133](#), [155](#), [166](#), [169](#), [178](#), [188](#), [198](#),  
[207](#), [212](#), [225](#), [229](#), [243](#), [252](#), [271](#), [282](#), [284](#)

Schema [300](#)

Score [142](#), [264](#), [297](#)

Search [57](#), [259](#), [292](#)

Season [2](#)

Seed [24](#), [164](#), [244](#), [261](#), [294](#)

Series [2](#)

Setting Up [53](#)

Shutting Down [86](#), [95](#), [188](#), [252](#), [270](#)

Size [134](#), [183](#), [229](#)

Solaris [74](#)

Sonic Attributes [6](#)

Sources [250](#)

SQLite [21](#), [47](#), [52](#), [73](#), [91](#), [115](#), [119](#), [123](#), [126](#), [133](#), [155](#), [166](#), [169](#), [227](#), [243](#), [250](#), [284](#)

Station [37](#), [267](#)

Status [104](#), [181](#), [195](#)

Storage [21](#), [24](#), [73](#), [79](#), [122-123](#), [133](#), [227](#), [238](#)

Submit [47](#), [51](#), [69](#), [91](#), [104](#), [164](#), [274-275](#), [279](#), [284-285](#)

## T

Tempo [6](#), [22-23](#), [125](#), [244](#), [255](#), [263](#), [296](#)

Testing [99](#), [272](#)

Text [11](#), [13](#), [46](#), [57](#), [79](#), [126](#), [142](#), [145](#), [167](#), [172](#), [219](#), [263](#), [296](#), [299](#)

Text Search [57](#), [299](#)

Third-Party

Identifiers [9](#)

Title [6-7](#), [38](#), [59](#), [76](#), [83](#), [143](#), [145](#), [155](#), [159](#), [167](#), [170](#), [173](#), [184](#), [198](#)

TOC [280](#)

Track [6](#), [13](#), [22-23](#), [38](#), [76](#), [139](#), [143](#), [149](#), [156](#), [162](#), [169](#), [173](#), [175](#), [203](#), [268](#), [273-275](#), [280-282](#), [285](#)

TrackID [15](#), [50](#), [67](#), [179](#), [188](#)

Tuning [269](#)

TVCHANNEL [230](#)

## U

User [37](#), [56](#), [72](#), [80](#), [89](#), [93](#), [107](#), [155](#), [167](#), [169](#), [175](#), [189](#), [198](#), [202-203](#), [243](#), [281](#), [283](#)

Handle [80](#)

## V

Validation [274](#)

Value Key [146](#)

Values [2](#), [27](#), [31](#), [110](#)

Video [2](#), [41](#), [104](#), [107](#), [142](#), [163](#)

Product [2](#), [42](#), [142](#), [163](#)

VideoID [91](#), [142](#)

Voice Commands [31](#)

## W

Windows [53](#), [71](#), [102](#)

Windows CE [73](#)

Work [2](#), [7](#), [42](#), [151](#)