GNSDK for Desktop Developers Guide

C API

(Gracenote Developer Portal Edition)

Release 3.07.7.3701

Published: 7/14/2015 12:46 PM

Gracenote, Inc.
2000 Powell Street, Suite 1500

Emeryville, California
94608-1804
www.gracenote.com

Preface

Start Here - Documentation Roadmap

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

- HTML5 Help
- PDF

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

Doc Set Summary

Document	Description	Location in Package
GNSDK for Desktop Help System	The most comprehensive format of GNSDK for Desktopdocumentation.	/docs/start_here.html
GNSDK for Desktop Developers Guide	*This document* Detailed information about developing GNSDK for Desktop applications.	docs/GNSDK-for-*-Developers-Guide-C-Language.pdf
GNSDK for Desktop Release Notes (*PDF only*)	Summary of what's new and changed in this release.	docs/GNSDK-for-*-Release-Notes.pdf

API Reference Documentation

In addition to several popular object-oriented languages, you can develop GNSDK for Desktop applications using C. The following table shows where to find the C API Reference documentation:

API Name	Description	Location in Package
C API Reference	API descriptions of the GNSDK for Desktop C interface.	docs/Content/api_ref_c/html/index.html

Sample and Reference Applications

Code Samples	Description	Location in Package
Sample applications	Applications you can compile and run that demonstrate specific GNSDK for Desktopfeatures.	/samples
Reference applications	Applications and source code that demonstrate several GNSDK for Desktopfeatures.	/reference_apps

Concepts

Introduction

About Gracenote

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

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

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

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

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

What is Gracenote SDK?

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

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

GNSDK for Desktop and Gracenote Products

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

Gracenote Products	GNSDK for Desktop Modules	Module Description
MusicID™	MusicID, MusicID-File	MusicID (CD TOC, Text, Stream and File Fingerprint); MusicID-File (LibraryID, AlbumID and TrackID Fingerprint Identification) Enables MusicID recognition for identifying CDs, digital music files, and streaming audio and delivers relevant metadata such as track titles, artist names, album names, and genres. Also provides library organization and direct lookup features.
VideoID™ and Video Explore™	Video	Provides video item recognition for DVD and Blu-ray products via TOCs. Provides extended video recognition and searching, enabling user exploration and discovery features.
Music Enrichment	Link	A mechanism for retrieving enriched content including cover art, artist images, biographies and reviews.
Playlist	Playlist, MoodGrid	Generates playlists based on music descriptors. Includes More Like This™ functionality to generate a playlist similar to the currently playing track. Also supports creation of MoodGrids to visually group content based on mood metadata.

Modules Overview

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

General GNSDK for Desktop Modules

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

Link: Optional.The Link module provides access to enriched metadata. To use this module, your application requires special metadata entitlements.

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

DSP: Optional. Provides digital signal processing functionality required for fingerprint generation.

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

Submit: Provides functionality necessary to submit metadata parcels to Gracenote database.

Product-Specific Modules

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

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

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

VideoID: Provides DVD and BluRay identification, and Text search for video; encompasses the VideoID and Video Explore products.

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

Gracenote MetaData

Gracenote Media Elements

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

Music

- Music CD
- Album
- Track
- Artist
- Contributor

Video

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

Music Terminology

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

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

Video Terminology

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

- **Video Work** - 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.
- **Video 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 (Univeral Product Code), Hinban, or EAN (European Article Number). Products are for the most part released on a physical format, such as a DVD or Blu-ray.
- **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 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).
- **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 Clip** - A short video presentation.
- **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.
- **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.
- **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.
- **Chapter** - A Video feature may contain chapters for easy navigation and as bookmarks for partial

viewing.

- **Season** - A Season is 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 Series is a collection of related Works, typically in sequence, and often comprised of Seasons (generally for a TV series), for example: CSI: Miami, CSI: Vegas, CSI: Crime Scene Investigation.

Genre and Other List-Dependent Values

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

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

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

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.

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 Global Services & Support representative for more information.

Below are examples of a Simplified and Detailed Genre Hierarchy Groups. The values below are for documentation purposes only.

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

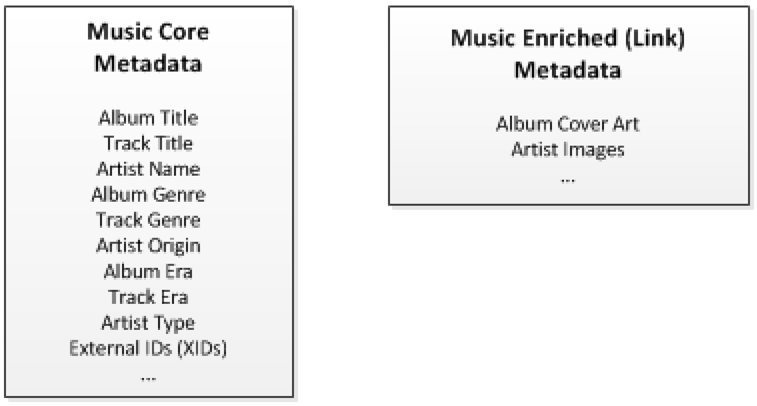
- Level-1: 25 genres
  - Level-2: 250 genres
    - Level-3: 800 genres

Contact your Gracenote representative for more detail.

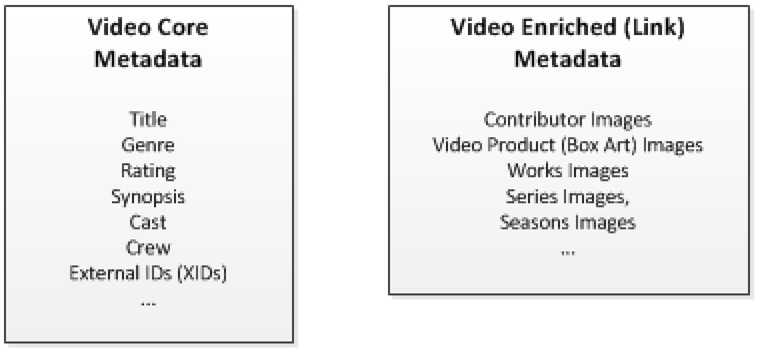
Core and Enriched Metadata

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

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



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



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 Global Services & Support representative for more information.

Related Tasks

Classical Music Metadata

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

Classical Three-Line Display: Metadata Mapping

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

Classical Three-Line Display: Metadata Mapping Example

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

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

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

Related Tasks

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.

Related Tasks

Clean-up and Collaborative Artists

Gracenote powers an easier navigation experience by utilizing Gracenote's clean metadata. Slight misspellings and nicknames can be cleaned-up into a single name for quicker, more accurate navigation. For example, tracks by "CCR", "C.C.R" and "Creedence Clearwater Revival" can all be cleaned up into a single "Creedence Clearwater Revival" entity.

Collaborations between two or more artists can often clutter an entire screen. For example, the Santana album "Supernatural" contains a number of collaborations such as "Santana featuring Rob Thomas" and "Santana featuring Dave Matthews" which would appear individually when navigating by Artist. Gracenote enhances the navigation experience with Collaborative Artists by providing the ability to consolidate by the Primary Artist. All of Santana's tracks, including collaborations, can be grouped under a single "Santana" artist making it much easier to navigate.

Related Tasks

Music Modules

Music Module Overview

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

MusicID Overview

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

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

- 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 for Desktop 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)

CD TOC Recognition

MusicID-CD is the component of GNSDK for Desktop that handles recognition of audio CDs and delivery of information including artist, title, and track names. The application provides GNSDK for Desktop 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.

Related Tasks

Text-Based Recognition

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

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

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

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

Related Tasks

Fingerprint-Based Recognition

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

Waveform Recognition

A fingerprint is generated from a short audio sample of about six seconds. The fingerprint is sent to Gracenote Services for identification. Accessing fingerprints requires an Internet connection, which must be provided by the application.

There are two basic types of fingerprints:

- MusicID (File fingerprint)
- MusicID (Stream)

MusicID (File fingerprint) recognizes audio files such as those generated when ripping a CD. It uses the first portion of the audio file for recognition.

MusicID (Stream) can identify music using short samples from anywhere within a song. Fingerprints can be generated from a 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 for Desktop modules. This module is optional unless the application performs music identification or generates audio features for submission to Gracenote.

Related Tasks

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.

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

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.

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 (e.g., fingerprinting, setting metadata, returned statuses, returned results, and so on.). The status or result callbacks provide the only mechanism for accessing Response GDOs.

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 appliction 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 Tkeep 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.

Related Tasks

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

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).

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.

MusicID-Stream

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

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

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

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

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

Radio Overview

Gracenote Radio allows you monitor audio streams, providing recognition and metadata retrieval for terrestrial, Internet, and satellite radio stations. Gracenote Radio is a subset of MusicID-Stream, and is intended for use without any end-user interaction.

Radio:

- Recognizes streaming audio from traditional radio sources, including AM/FM, HD, DAB, and others.
- Recognizes music automatically without the need to start or stop recognition.
- Supports broadcast metadata such as title, artist, or station.

An application can use the information that Radio provides to display the song's metadata and imagery (if licensed), or in combination with other Gracenote software to generate playlists or other specialized applications based on the radio listening habits of the end-user.

Gracenote Radio requires the following in order to function optimally:

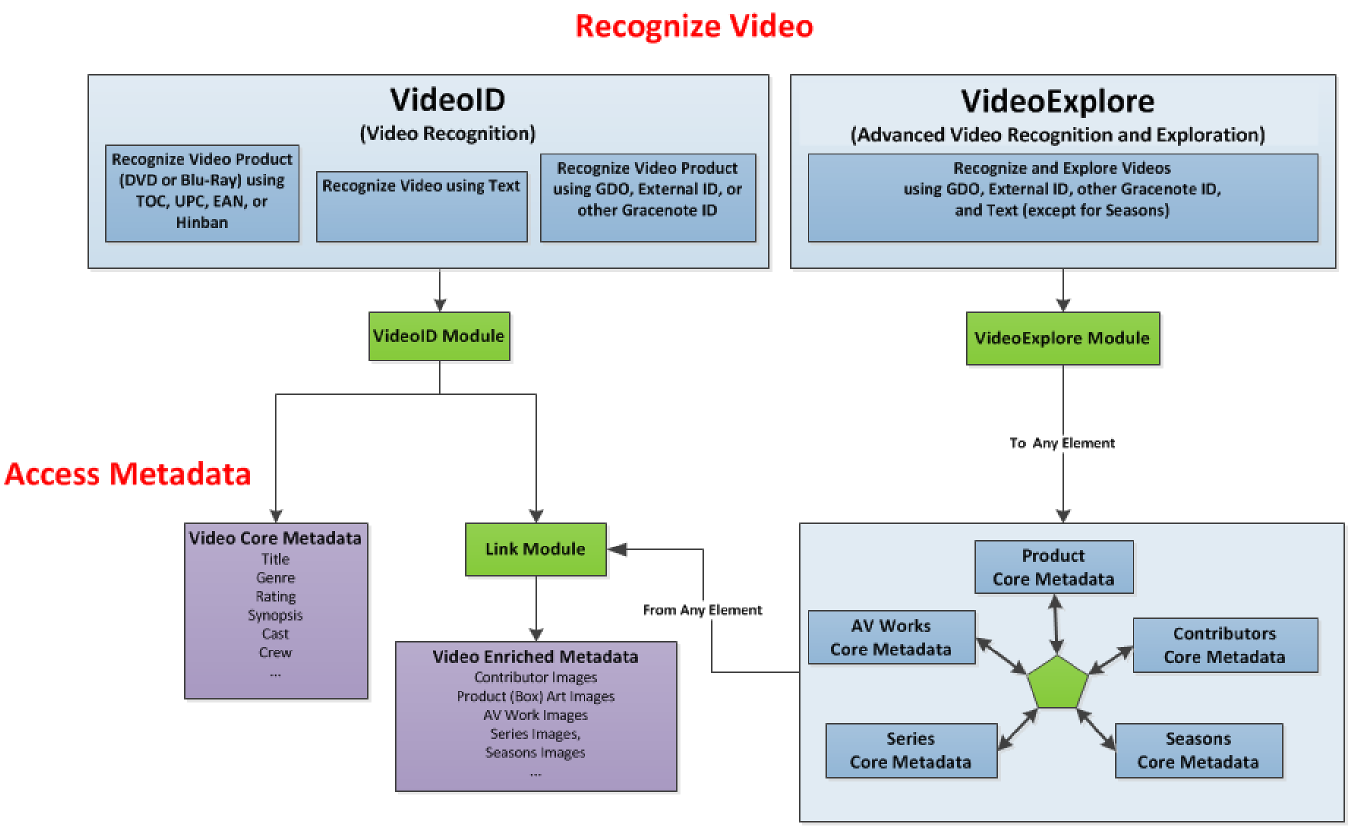
- Clean audio (such as line-in and other audio sources that are not from a microphone)
- Any available broadcast metadata (such as RDS or shoutcast metadata).
- Specific application interactions (such as station changes).

Video Modules

Video Module Overview

GNSDK provides two modules for implementing video features: VideoID and VideoExplore:

- VideoID supports recognizing DVDs and Blu-ray discs, and enables access to related metadata such as title, genre, rating, synopsis, cast and crew.
- Video Explore enables access to additional video elements and their metadata, visual assets, trailers, and commerce links.



VideoID Overview

VideoID can recognize Products (DVDs/Blu-Ray discs) using any of the following as input:

- TOC (table of contents)
- Text (a Product title)

- External ID, such as UPC, EAN, and Hinban codes
- GDO other Gracenote ID for the Product, such as a GNID, or TUI/TUI Tag combination.

VideoID applications support these features:

- Recognize a Product using any of the input types listed above.
- Enable access to core metadata from a Product GDO
- Enable access to enriched metadata from a Product GDO using the Link module

Related Tasks

VideoExplore Overview

Using VideoExplore, an application can perform advanced video recognition and exploration using any of the following as input:

- Text, such as Product and AV Work titles, and names of Series and Contributors.
- External ID, such as UPC, EAN, and Hinban codes
- GDO other Gracenote ID for the Product, such as a GNID, or TUI/TUI Tag combination.

VideoExplore applications support these features:

- Recognize Products, AV Works, Contributors, Series, and Seasons using the above inputs. Note: Text is not supported for Seasons. Seasons can be retrieved using GDOs returned through other GNSDK queries
- Enable access to core metadata from any video GDO
- Enable access to enriched metadata from any video GDO using the Link module
- Perform suggestion searches for titles of Product, AV Work, Contributors, and Series. Request and receive suggestions for matching Contributors, Series and AV Works using user Text inputs. Suggestion searches enables applications to minimize the number of character inputs necessary for a user to find a match. GNSDK will return suggestions, if available, using a GDO.
- Navigate from one video element to a related video elements and access their core metadata.

Related Tasks

Discovery Features

Playlists

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

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

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

Collection Summaries

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

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

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

More Like This

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

Related Tasks

Implementing Discovery Features........170

.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 for Desktop's MusicID or MusicID-File (or both) to recognize music media and create valid GDOs.

- The application uses valid unique identifiers. A unique identifier must be a valid UTF-8 string of unique media identifier data. For more information, see "Unique Identifiers" on page 22.

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

Playlist Storage Recommendations

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

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

Playlist Resource Requirements

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

Use Case	Typical Scenario	Number of Collection Summaries	Application Provides Collection Summary to Playlist	Required Computing Resources
Single user	Desktop user Mobile device user	Generally only one	Once, normally at start-up	Minimal-to-average, especially as data is ingested only once or infrequently
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 8. As such, Playlist performs certain behaviors when evaluating tracks using hierarchical attribute criteria.

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

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

Key Playlist Components

Playlist operates on several key components. The GNSDK for Desktop 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 Desktop for operation, and consequently must be implemented with one or both of GNSDK for Desktop'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. The identifier is a value that allows the application to identify the physical media being referenced. The identifier is not interpreted by Playlist; it is only returned to the application in Playlist results. An identifier is generally application-dependent. For example, a desktop application typically uses a full path to a file name for an identifier, while an online application typically uses a database key. The media GDO should contain relevant metadata for the media referenced by the identifier. In most cases the GDO comes from a recognition event for the respective media (such as from MusicID). Playlist will take whatever metadata is relevant for playlist generation from the given GDO. For best results, Gracenote recommends giving Album Type GDOs that have matched tracks to this function; Track Type GDOs also work well. Other GDOs are supported, but most other types lack information for good Playlist generation.

Collection Summary

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

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

Storage

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

Seed

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

Playlist generation

GNSDK for Desktop 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().

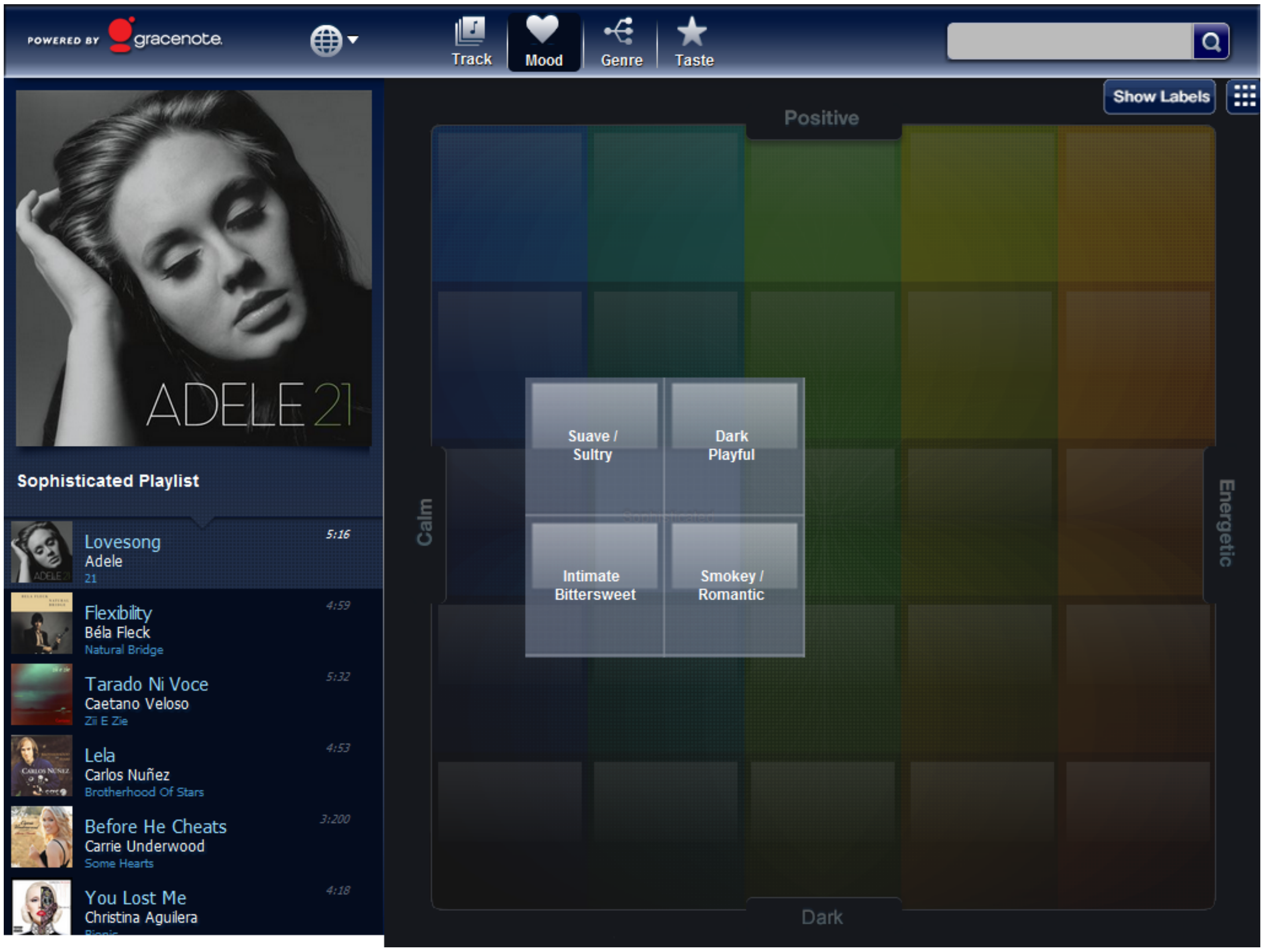
NOTE: Gracenote recommends streamlining your Playlist implementation by using the provided More Like This™ function, gnsdk_playlist_generate_morelikethis(), which uses the More Like This algorithm to generate optimal playlist results and eliminates the need to create and validate PDL statements.

MoodGrid Overview

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

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

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


POWERED BY gracenote
Track
Mood
Genre
Taste
Show Labels
Positive
Calm
Energetic
Dark
Suave / Sultry
Dark Playful
Intimate Bittersweet
Smokey / Romantic
ADELE 21
Sophisticated Playlist
Lovesong Adele 21 5:16
Flexibility Béla Fleck Natural Bridge 4:59
Tarado Ni Voce Caetano Veloso Zii E Zie 5:32
Lela Carlos Nuñez Brotherhood Of Stars 4:53
Before He Cheats Carrie Underwood Some Hearts 3:200
You Lost Me Christina Aguilera 4:18

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

Related Tasks

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. 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

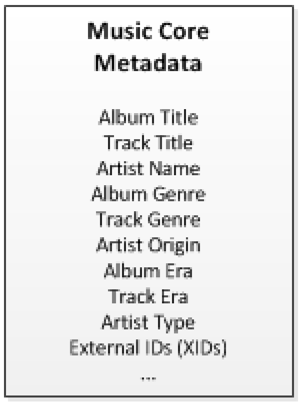
When feedback is sent, Rhythm reconfigures the station’s playlist accordingly. Rhythm supports the retrieval of the playlist at any time before or after feedback is provided.

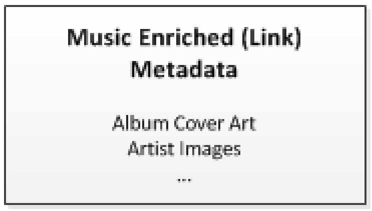
A station's behavior may also be tuned by changing playlist generation behavior. For example, you can get difference responses based on options for popularity or similarity.

Enriched Content Module (Link)

Link Module Overview

The Link module allows applications to access *enriched content* beyond standard core metadata. The diagram below show the difference between core metadata and enriched content for music.



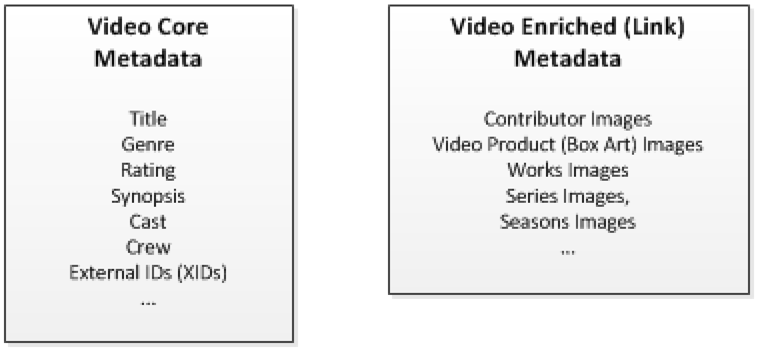


Link allows applications to access and present enriched content related to media that has been identified using GNSDK identification features. Link delivers third-party content identifiers matched to the identified media, which can then be used to retrieve enriched content from Gracenote. Link can also return any custom information that a customer has linked to Gracenote data—such as their own media identifiers.

Without Link, developers themselves must match and integrate data from various sources to present a targeted, relevant content experience to end-users during music playback. By providing developers with the ability to retrieve enriched content, Gracenote enables its customers to generate new revenue streams while improving the music listening experience.

Enriched content offered through Gracenote Link includes:

- Music enrichment:
  - Album cover art
  - Artist images
  - Album reviews
  - Artist biographies
- Third-party IDs



Related Tasks

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.

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.

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 Global Services & Support 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.

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.

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.

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.

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.

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

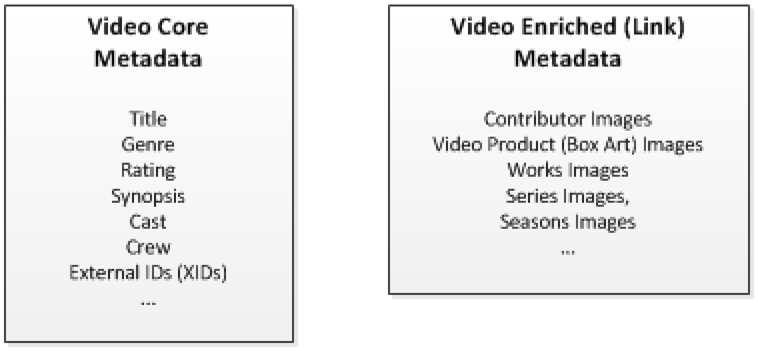
Video Contributor images typically conform to a 6:9 (width:height) aspect ratio. Two ranges are provided due to larger variation in image dimensions.

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

Video Enrichment Overview

All Gracenote customers can access core metadata products they license. Optionally, customers can access additional metadata, known as enriched metadata by purchasing additional metadata entitlements. Applications generally access enriched metadata using the Link module APIs. Enriched metadata includes Product box art, images, video trailers, and others.


Video Core Metadata
Title
Genre
Rating
Synopsis
Cast
Crew
External IDs (XIDs)
...
Video Enriched (Link) Metadata
Contributor Images
Video Product (Box Art) Images
Works Images
Series Images,
Seasons Images
...

Supported Platforms and System Requirements

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

Gracenote Developer Portal System Requirements

Platform	Architecture	Bit	S/S	OS	Compiler/Note
Android	armeabi-v7a	32-bit	shared		gcc 4.6
iOS	armv7	32-bit static		iOS 4.3 and above	gcc 4.2.1
	armv7s				
	x86				
Linux	arm	32-bit	shared		gcc 4.5.1
	mips	32-bit	static		gcc 4.5.2
	x86	32-bit	shared	2.6 kernel and above	gcc 3.4.6
		64-bit	shared		
MacOS	x86	32-bit	shared	OSX 10.4 and above	gcc 4.2.1
		64-bit			
Windows	x86	32-bit	shared	Windows XP or above	MS Visual studio 2012
		64-bit			
Winphone8	ARM	32-bit	shared		MS Visual studio 2012
	x86				
Winrt	ARM	32-bit	shared		MS Visual studio 2012
	x86				

*The GNU C Library (glibc), revision 2.3.2 and newer, is typically available on Linux distributions released during or after 2003.

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

Memory Usage

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

Memory Usage Guidelines

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

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

RAM Requirements

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

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

Feature Set	**Measured RAM Requirements***	**Recommended RAM Requirements***
MusicID (CD, Text) Music Enrichment (Cover Art)	3 MB	8 MB

Feature Set	Measured RAM Requirements*	Recommended RAM Requirements*
MusicID (CD, Text) Music Enrichment (Cover Art) MusicID (File) Playlist, MoodGrid (10K Tracks)	6 MB	14 MB
MusicID (CD, Text) Music Enrichment (Cover Art) MusicID (File) Playlist, MoodGrid (20K Tracks)	9 MB	20 MB
MusicID (CD, Text) Music Enrichment (Cover Art) MusicID (File) Playlist, MoodGrid (40K Tracks)	12 MB	32 MB

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

Playlist Memory Usage

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

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

Development Library Sizes

The following table lists the library sizes for GNSDK for Desktop.

Desktop 3.x

Platform	Architecture	Type	Size	Notes

Android	armeabi	Dynamic	1 M	Utilities
	armeabi	Dynamic	11 M	
	armeabi-v7a	Dynamic	1 M	Utilities
	armeabi-v7a	Dynamic	11 M	
	x86	Dynamic	1 M	Utilities
	x86	Dynamic	11 M	
iOS	armv7-32	Static	8 M	
	armv7s-32	Static	8 M	
	x86-32	Static	8 M	
Linux	arm-32	Dynamic	1 M	Utilities
	arm-32	Dynamic	11 M	
	arm-32	Static	2 M	Utilities
	arm-32	Static	7 M	
	armhf-32	Dynamic	1 M	Utilities
	armhf-32	Dynamic	9 M	
	armhf-32	Static	2 M	Utilities
	armhf-32	Static	6 M	
	mips-32EL	Static	8 M	
	x86-32	Dynamic	168 M	Optional IPP Library
	x86-32	Dynamic	6 M	Utilities
	x86-32	Dynamic	15 M	
	x86-32	Static	5 M	Utilities
	x86-32	Static	10 M	
	x86-64	Dynamic	207 M	Optional IPP Library
	x86-64	Dynamic	6 M	Utilities
	x86-64	Dynamic	17 M	
	x86-64	Static	6 M	Utilities
	x86-64	Static	13 M	

MacOS	x86-32	Dynamic	2 M	Utilities
	x86-32	Dynamic	8 M	
	x86-32	Static	2 M	Utilities
	x86-64	Dynamic	3 M	Utilities
	x86-64	Dynamic	10 M	
	x86-64	Static	3 M	Utilities
Windows	x86-32	Dynamic	149 M	Optional IPP Library
	x86-32	Dynamic	16 M	Utilities
	x86-32	Dynamic	53 M	
	x86-32	Static	40 M	Utilities
	x86-64	Dynamic	16 M	Utilities
	x86-64	Dynamic	53 M	
	x86-64	Static	49 M	Utilities

where (M = 1024^2 bytes).

Getting Started

GNSDK C Quick Start Tutorial

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.

Setting Up Your Development Environment

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

Include Files

Include files for GNSDK applications are located in `/include` under your installation directory. There are approximately 25 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. You should not need to include anything other than `gnsdk.h` in your application for GNSDK.

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 13 (Windows has 26, including DLLs) library files containing the binaries for the 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 Global Services & Support.

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, etc.) 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";   // client app version - usually "1"
    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 intializations 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.

All the GNSDK functions return "GNSDK_SUCCESS" if completed successfuly. 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(&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);
    }
     /* Initialize User object */
    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, "GDO Workflows".

```
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_matches(query_handle, &response_gdo)) != GNSDK_
SUCCESS)
        {
            _display_error(__LINE__, "gnsdk_musicid_query_find_matches()", error);
       }
       /* Find number of matches */
       else if ((error = gnsdk_manager_gdo_child_count(response_gdo, GNSDK_GDO_CHILD_MATCH, &count)) !=
GNSDK_SUCCESS)
       {
           _display_error(__LINE__, "gnsdk_manager_gdo_child_count(GNSDK_GDO_CHILD_MATCH)", 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_MATCH, 1, &album_gdo))
!= GNSDK_SUCCESS)
       {
            _display_error(__LINE__, "gnsdk_manager_gdo_child_get(GNSDK_GDO_CHILD_MATCH)", 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_str_t   updated_serialized_user_string  = GNSDK_NULL;
    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_musicid_shutdown();
    gnsdk_manager_shutdown();
}
```

Conclusion

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

```
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 the SDK to see how to search for different types of metadata by audio fingerprint, text, or existing file.

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);
    }
       /* Initialize User object */
       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_str_t   updated_serialized_user_string  = GNSDK_NULL;
    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_musicid_shutdown();
    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_matches(query_handle, &response_gdo)) != GNSDK_
SUCCESS)
           {
               _display_error(__LINE__, "gnsdk_musicid_query_find_matches()", error);
           }
               /* Find number of matches */
               else if ((error = gnsdk_manager_gdo_child_count(response_gdo, GNSDK_GDO_CHILD_MATCH, &count)) !=
GNSDK_SUCCESS)
               {
                       _display_error(__LINE__, "gnsdk_manager_gdo_child_count(GNSDK_GDO_CHILD_MATCH)", 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_MATCH, 1, &album_gdo))
!= GNSDK_SUCCESS)
               {
                       _display_error(__LINE__, "gnsdk_manager_gdo_child_get(GNSDK_GDO_CHILD_MATCH)", 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);
}

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);
}
```

Using the Sample Applications

GNSDK for Desktop 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.

Sample Applications and Reference Applications

Below is a list of sample applications and reference applications for GNSDK for Desktop. 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, etc., 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 intializations 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, etc. 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	Modules Used	Description	Location in Package
moodgrid	Playlist	Provides a complete MoodGrid 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

Name	Modules Used	Description	Location in Package
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
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

Name	Modules Used	Description	Location in Package
musicid_lookup_album_id	MusicID	Lookd up an Album based on its TOC.	samples/musicid_ lookup_album_id/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	MusicID	Uses MusicID-Stream to fingerprint and identify a music track.	samples/musicid_ stream/main.c
playlist	Playlist	Demonstrates using the Playlist APIs to create More Like This and custom playlists.	samples/playlist/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

Reference Applications

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

Building a Sample Application

GNSDK for Desktop provides sample applications for many common use cases. You can find the sample applications in the samples/<module> folders of your GNSDK for Desktop package, where <module> corresponds to GNSDK for Desktop 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 for Desktop 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 for Desktop supported platforms. For more information about supported platforms, see "Supported Platforms and System Requirements" on page 31.

Build Environment Requirements

Building the sample applications requires the following environment:

- GNU Make 3.81 or above
- A command-line environment supported by the target platform

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

Including Client and License Information to Run a Sample

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"
```

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.

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 50.
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 for Desktop 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 50.

MacOS

Follow the general instructions in "Build Steps" on page 50.

To build samples when using Mac OS X x86_64 binaries, use the command:

make ARCH=x86_64

MusicID Sample Application Walkthrough

GNSDK MusicID Sample Application

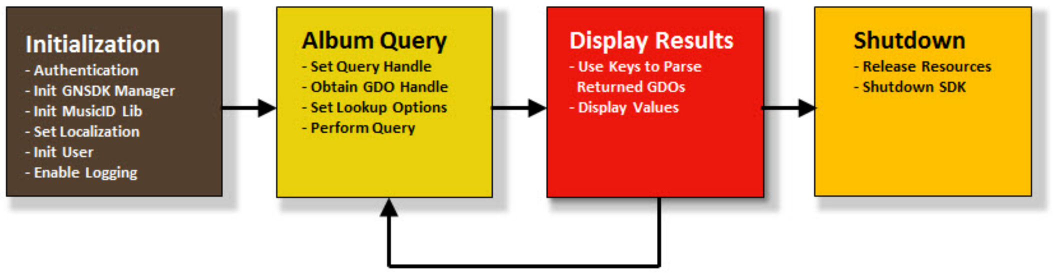
This topic steps through the GNSDK for Desktop sample application called musicid_gdo_navigation.

Sample Application: musicid_gdo_navigation/main.c

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 for Desktop 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.

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 Gynth 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]
```

Prerequisites

To build and run any of the sample applications, see "Building a Sample Application" on page 50. In general, you need three things from Gracenote:

1. License File—Details your permissions and access to GNSDK for Desktop libraries.
2. Client ID—Your specific GNSDK for Desktop 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 for Desktop libraries. You pass them when you invoke a sample application program at the command-line:

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

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 for Desktop 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, etc. */
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

The following two calls initialize the application's interaction with local storage. Both take the GNSDK Manager handle as a parameter. The SDK uses storage to cache queries (which improves online performance) and to power local lookups. Currently, SQLite is the only database provided for local storage and requires initialization. This must be done before the local lookup initialization call.

```
/*
 * Initialize the Storage SQLite Library
 */
gnsdk_storage_sqlite_initialize(sdkmgr_handle);

/*
 * If you wish to do local lookups, initialize the storage folder (/sample_db) for Local Lookup
 */
gnsdk_storage_sqlite_option_set(GNSDK_STORAGE_SQLITE_OPTION_STORAGE_FOLDER, "../sample_db");

/*
 * Initialize the Lookup Local Library
 */
gnsdk_lookup_local_initialize(sdkmgr_handle);
```

Initialize the MusicID Library Manager

Next, the application then initializes its interaction with the Gracenote MusicID library. Every GNSDK for Desktop 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 for Desktop by their handles. These handles contain the Client ID 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 onine 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.

MusicID 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 11.

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 82.

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 203.

This sample application (musicid_gdo_navigation) uses unique identifiers called TUIs to perform the initial identification (lookup) query. TUI stands for "Title Unique Identfier" 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 for Desktop 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 82. 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_musicid_query_set_gdo(query_handle, album_gdo);
gnsdk_manager_gdo_release(response_gdo);

/*
 * Query for this match in full
 */
gnsdk_musicid_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, addtional 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.

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 for Desktop:

```
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);
```

Implementing Applications

Basic Application Design Tasks

Application Flow

This topic describes how to get started with GNSDK for Desktop development.

Every GNSDK for Desktop application follows this general design and application flow:

1. Include the GNSDK for Desktop header files for the modules your application requires. These should be based on the Gracenote products you licensed, such as MusicID, or MusicID-File.
2. Include other application headers that your application requires.
3. Initialize GNSDK 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 for Desktop 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 82.
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 GNSDK for Desktop the GNSDK Manager.

Setup and Initialization

Working with Header Files and Libraries

GNSDK for Desktop 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 for Desktop 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 SDK
- **GNSDK_LINK:** Interface for Link content
- **GNSDK_LOOKUP_LOCAL:** Interface for the Lookup Local module
- **GNSDK_LOOKUP_LOCALSTREAM:** Interface for the Lookup Local Stream module
- **GNSDK_MANAGER:** Interface for the GNSDK Manager module
- **GNSDK_MOODGRID:** Interface for Mood Grid
- **GNSDK_MUSICID:** Interface for MusicID
- **GNSDK_MUSICID_FILE:** Interface for MusicID File
- **GNSDK_MUSICID_STREAM:** Interface for MusicID Stream
- **GNSDK_PLAYLIST:** Interface for the Playlist SDK
- **GNSDK_STORAGE_SQLITE:** Interface for the SQLite SDK
- **GNSDK_SUBMIT:** Interface for the Submit SDK
- **GNSDK_VIDEO:** Interface for VideoID

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.

Getting Manifest Information about Local Databases

GNSDK for Desktop 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 versions and available image sizes, use the following functions:

- gnsdk_lookup_local_storage_info_get(): Returns information for a particular local database and local storage key
- gnsdk_lookup_local_storage_info_count(): Returns a count of the values available for a particular local database and local storage key

Images

GNSDK for Desktop provides album cover art, and artist and genre images in different sizes. You can retrieve the available image sizes by using the GNSDK_LOOKUP_LOCAL_STORAGE_IMAGE_SIZE key with the gnsdk_lookup_local_storage_info_get() function. This allows you to request images in available sizes only, rather than spending cycles requesting image sizes that are not available.

Use the gnsdk_lookup_local_storage_info_count() function to provide ordinals to the gnsdk_lookup_local_storage_info_get() function to get the image sizes.

For example, the following code retrieves the available image sizes:

```
gnsdk_uint32_t count=0;
gnsdk_uint32_t ordinal=0;
gnsdk_cstr_t available_image_size = GNSDK_NULL;
gnsdk_error_t error = GNSDK_SUCCESS;

error = gnsdk_lookup_local_storage_info_count(
                       GNSDK_LOOKUP_LOCAL_STORAGE_CONTENT,
                       GNSDK_LOOKUP_LOCAL_STORAGE_IMAGE_SIZE, &count);
if (!error)
{
    for (ordinal=1; ordinal<=count; ordinal++)
    {
        error = gnsdk_lookup_local_storage_info_get(
                       GNSDK_LOOKUP_LOCAL_STORAGE_CONTENT,
                       GNSDK_LOOKUP_LOCAL_STORAGE_IMAGE_SIZE,
                       ordinal, &available_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 Music Content" on page 165.

Local Database Versions

To retrieve the version number for a local database, use the GNSDK_LOOKUP_LOCAL_STORAGE_GDB_VERSION key with the gnsdk_lookup_local_storage_info_get() function. Use an ordinal of 1 to get the database version.

For example, the following code retrieves the local database versions:

```
gnsdk_cstr_t version = GNSDK_NULL;
gnsdk_error_t error = GNSDK_SUCCESS;

error = gnsdk_lookup_local_storage_info_get(
                       GNSDK_LOOKUP_LOCAL_STORAGE_CONTENT,
                       GNSDK_LOOKUP_LOCAL_STORAGE_GDB_VERSION, 1, &version);
if (!error)
{
    error = gnsdk_lookup_local_storage_info_get(
                       GNSDK_LOOKUP_LOCAL_STORAGE_METADATA,
                       GNSDK_LOOKUP_LOCAL_STORAGE_GDB_VERSION, 1, &version);
}
if (!error)
{
    error = gnsdk_lookup_local_storage_info_get(
                       GNSDK_LOOKUP_LOCAL_STORAGE_TOCINDEX,
                       GNSDK_LOOKUP_LOCAL_STORAGE_GDB_VERSION, 1, &version);
}
if (!error)
{
    error = gnsdk_lookup_local_storage_info_get(
                       GNSDK_LOOKUP_LOCAL_STORAGE_TEXTINDEX,
                              GNSDK_LOOKUP_LOCAL_STORAGE_GDB_VERSION, 1, &version);
}
```

Locales

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 99.

Authorizing a GNSDK for Desktop Application

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

As a Gracenote customer, Gracenote Global Services & Support works with you to determine the kind of products you need (such as MusicID, Playlist, and so on). Gracenote also determines the 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 for Desktop 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 for Desktop 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 Global Services & Support representative for more information.

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

License Files

Gracenote provides a license file along with your client ID. The license file notifies Gracenote to enable the GNSDK for Desktop 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 for Desktop modules.

```
-- BEGIN LICENSE v1.0 ABC123XYZ --
                       licensee: Gracenote, Inc.
                       name: CustomerA
                       notes: [CustomerA Account#]

                       client_id: 999999
                       musicid_text: enabled
                       musicid_cd: enabled
                       playlist: enabled
                       local_images: enabled
                       local_mood: enabled

                       -- SIGNATURE ABC123XYZ --
                       ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890abcedefghijklmnopqrstuvwxyz
                       -- END LICENSE ABC123XYZ --
```

Client IDs

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

A client ID string has the following format: 123456-12345678901234567. It consists of a six-digit client ID, and a 17-digit client ID Tag, separated by a hyphen (-).

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.

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 string. This ensures that each application instance is receiving all required metadata entitlements.

Users are represented in GNSDK for Desktop 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 page 69.

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 for Desktop 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 */
}
```

Initializing and Shutting Down GNSDK

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.

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.

Shutting Down an Application

Calls to the initialize API on any GNSDK for Desktop 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.

Example: Initializing and Shutting Down

Sample Application: musicid_lookup_album_text/main.c

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

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.

Proxy Server Options

GNSDK for Desktop 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);
```

Network Options

GNSDK for Desktop 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 queriess. 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.

Testing Connection Ability

You can test your applications ability to conncect 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.

Configuring Logging

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

1. **Enable GNSDK logging** - This creates log file(s) that you and Gracenote Global Services & Support can use to evaluate and debug any problems your application might encounter when using the SDK.
2. **Enable GNSDK logging and add to it** - Use the *_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 for Desktop logging is to configure a single log file to capture all logged messages and errors from both your application and the SDK.

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 asychronously. 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 for Desktop 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.

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.

Implementing Callback Logging

You also have the option to direct GNSDK for Desktop 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() */
```

Online and Local Database Lookups

Working with Local Databases

Working with local databases is very similar to working with online Gracenote services. In both cases, queries can be set up in the same way, and results can be processed in the same way. To use a local database, you must initialize the local database and you must set the user lookup option to indicate that the local database should be used for all lookups.

Initializing a Local Database

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

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

Setting Storage Folder Paths

After successful initialization, Gracenote recommends immediately setting a valid storage folder path for local storage.

You can specify the location of all the SQLite database files at once, by setting a storage folder path using the GNSDK_STORAGE_SQLITE_OPTION_STORAGE_FOLDER option with the gnsdk_storage_sqlite_option_set() function.

You can also specify the location of individual storage files, by using gnsdk_lookup_local_storage_location_set() with the following storage IDs:

- GNSDK_LOOKUP_LOCAL_STORAGE_CONTENT (sets the storage folder path for gn_cds.gdb)
- GNSDK_LOOKUP_LOCAL_STORAGE_METADATA (sets the storage folder path for gn_mdata.gdb)
- GNSDK_LOOKUP_LOCAL_STORAGE_TEXTINDEX (sets the storage folder path for gn_itxt.gdb)
- GNSDK_LOOKUP_LOCAL_STORAGE_TOCINDEX (sets the storage folder path for gn_itoc.gdb)

Setting Local Lookup Mode

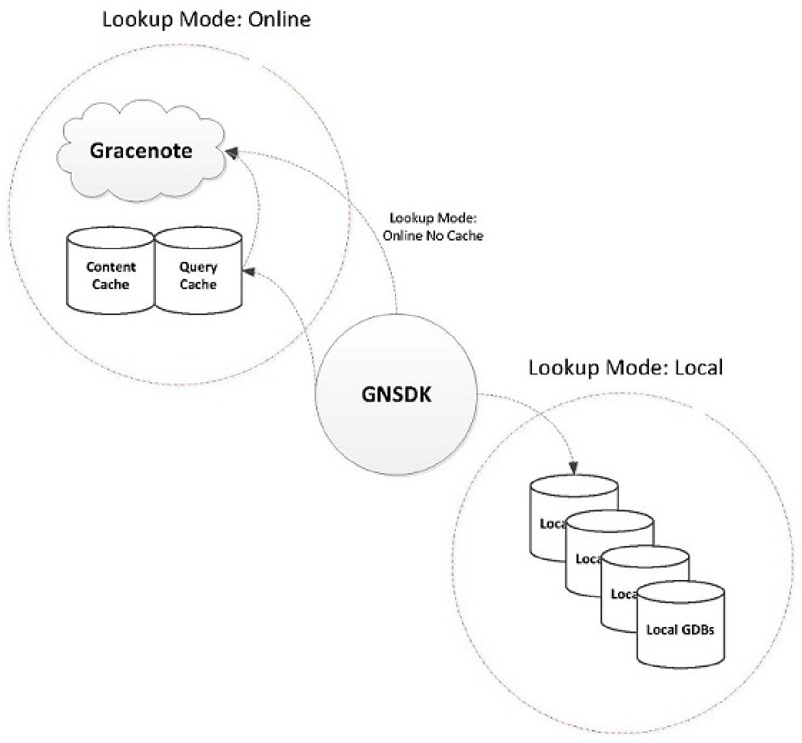
To do local lookups, you must set the user lookup option to indicate that the local database should be used for all lookups. For more information about the user lookup option, see "Setting Local and Online Lookup

Modes" on page 78.

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

Setting Local and Online Lookup Modes

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



Lookup Providers

A *lookup provider* is a module that implements the ability to query data for matches. GNSDK for Desktop 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 for Desktop 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.

Storage Providers

A *storage provider* is a module that implements storage for the entire SDK. GNSDK for Desktop allows various storage providers to be enabled at run-time. There are currently two storage providers available in GNSDK for Desktop: one implemented using SQLite, and one using QNX's QDB.

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.

Lookup Modes

GNSDK for Desktop allows you to determine whether lookups will be done locally or online, by setting *lookup modes*. Lookup modes affect the following types of local storage:

- Local databases, which are pre-populated to support fast local queries of the most popular items
- The online cache, which is a local cache for online-query results
- The lists store, which stores locale and list information and can be pre-populated or downloaded from the Gracenote Service

The lookup mode options allow an application to switch between the two main lookup providers (local and online). GNSDK for Desktop is designed to operate exactly the same way in either mode, providing identical behavior whether operating locally or online.

GNSDK for Desktop supports the following lookup mode options:

- GNSDK_LOOKUP_MODE_LOCAL
- GNSDK_LOOKUP_MODE_ONLINE
- GNSDK_LOOKUP_MODE_ONLINE_NOCACHE
- GNSDK_LOOKUP_MODE_ONLINE_NOCACHEREAD
- GNSDK_LOOKUP_MODE_ONLINE_CACHEONLY

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.

The local and online modes are the standard modes for applications to choose between. The other online options (NOCACHE, NOCACHEREAD, and CACHEONLY) are variations of the online mode. These additional online lookup modes give more control over when the SDK is allowed to perform a network connection and how to use the online-query cache. The online-query cache is used as a performance aid for online queries. If no storage provider is present, no online-query cache will be utilized.

NOTE: The lists cache is used as both an online-query cache and a local database for locale and list loading requests.

Lookup Mode Behavior

The following sections show the different lookup modes and their behavior:

GNSDK_LOOKUP_MODE_LOCAL

Queries are done using a local lookup provider.

- If a storage provider is enabled
  - If the storage provider is misconfigured, an error occurs and the query fails
  - The local lookup database is searched
- If no storage provider is enabled
  - An error occurs and the query fails

GNSDK_LOOKUP_MODE_ONLINE

Queries are done using an online lookup provider.

- If a storage provider is enabled
  - If the storage provider is misconfigured, an error occurs and the query fails
  - The online-query cache is searched
    - If the online-query search returns no results, a network query is performed
    - The results of the network query are written to the online-query cache
- If no storage provider is enabled
  - A network query is performed

GNSDK_LOOKUP_MODE_ONLINE_NOCACHE

Queries are done using an online lookup provider.

- If a storage provider is enabled
  - The online-query cache is ignored
  - A network query is performed
- If no storage provider is enabled
  - A network query is performed

GNSDK_LOOKUP_MODE_ONLINE_NOCACHEREAD

Queries are done using an online lookup provider.

- If a storage provider is enabled
  - If the storage provider is misconfigured, an error occurs and the query fails
  - A network query is performed
    - The results of the network query are written to the online-query cache

- If no storage provider is enabled
  - A network query is performed

GNSDK_LOOKUP_MODE_ONLINE_CACHEONLY

Queries are done using an online lookup provider.

- If a storage provider is enabled
  - If the storage provider is misconfigured, an error occurs and the query fails
  - The online-query cache is searched
- If no storage provider is enabled
  - An error occurs and the query fails

NOTE: Queries using GNFPX fingerprints are not cached locally.

Default Lookup Mode

If the application doesn’t set a lookup mode option, GNSDK for Desktop sets a default lookup mode. The default is GNSDK_LOOKUP_MODE_ONLINE, unless the GNSDK for Desktop license file limits all queries to be local-only, which prevents the SDK from connecting online. When this limit is set in the license file, the lookup mode defaults to GNSDK_LOOKUP_MODE_LOCAL.

Overriding the Lookup Mode for a Specific Query

You can set the lookup mode as a user option or set it separately for individual queries as an option on the query handle. Setting the GNSDK_USER_OPTION_LOOKUP_MODE option for a user handle applies to *all queries using the user handle*. You can override this for a specific query by setting the equivalent *query handle option*.

For example, you can override the setting for a music query by setting the GNSDK_MUSICID_OPTION_LOOKUP_MODE option. The query handle option uses the same option value keys as the user handle option.

Using Both Local and Online Lookup Modes

Your application can switch between local and online lookups, as needed. For example, the following pseudocode shows how to do a local Album TOC lookup using MusicID APIs, followed by an online lookup for cover art using Link APIs:

```
/* Local TOC lookup using MusicID
musicid_option_set(MODE_LOCAL)
musicid_toc_set(toc)
musicid_find_album(&album_gdo)
/* Online cover art lookup using Link
link_option_set(MODE_ONLINE)
link_set_gdo(album_gdo)
link_retrieve_content(cover)
```

Using Gracenote Data Objects

About Gracenote Data Objects (GDOs)

The primary goal of any GNSDK for Desktop 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 for Desktop 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 for Desktop – 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.

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.

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.

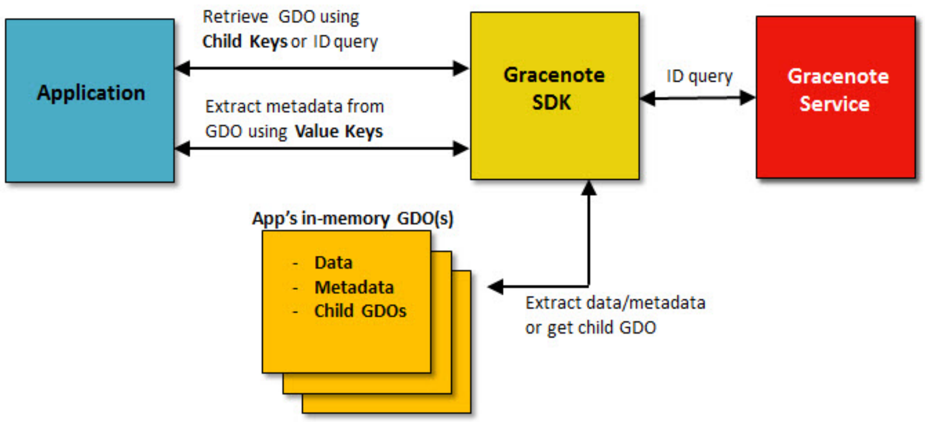
For example, an Album response GDO can contain child GDOs of type Track, or Artist, other Albums, and so on. A child GDO can contain its own child GDOs, such as Tracks, Artists, or Contributors, and so on.

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.

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.



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 (e.g., 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 for Desktop C API Reference*.

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.

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 for Desktop 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.

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.

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

Match Type	Description	Decision Needed?
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

Related Tasks

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.

GDO Workflow for a Single GDO Match

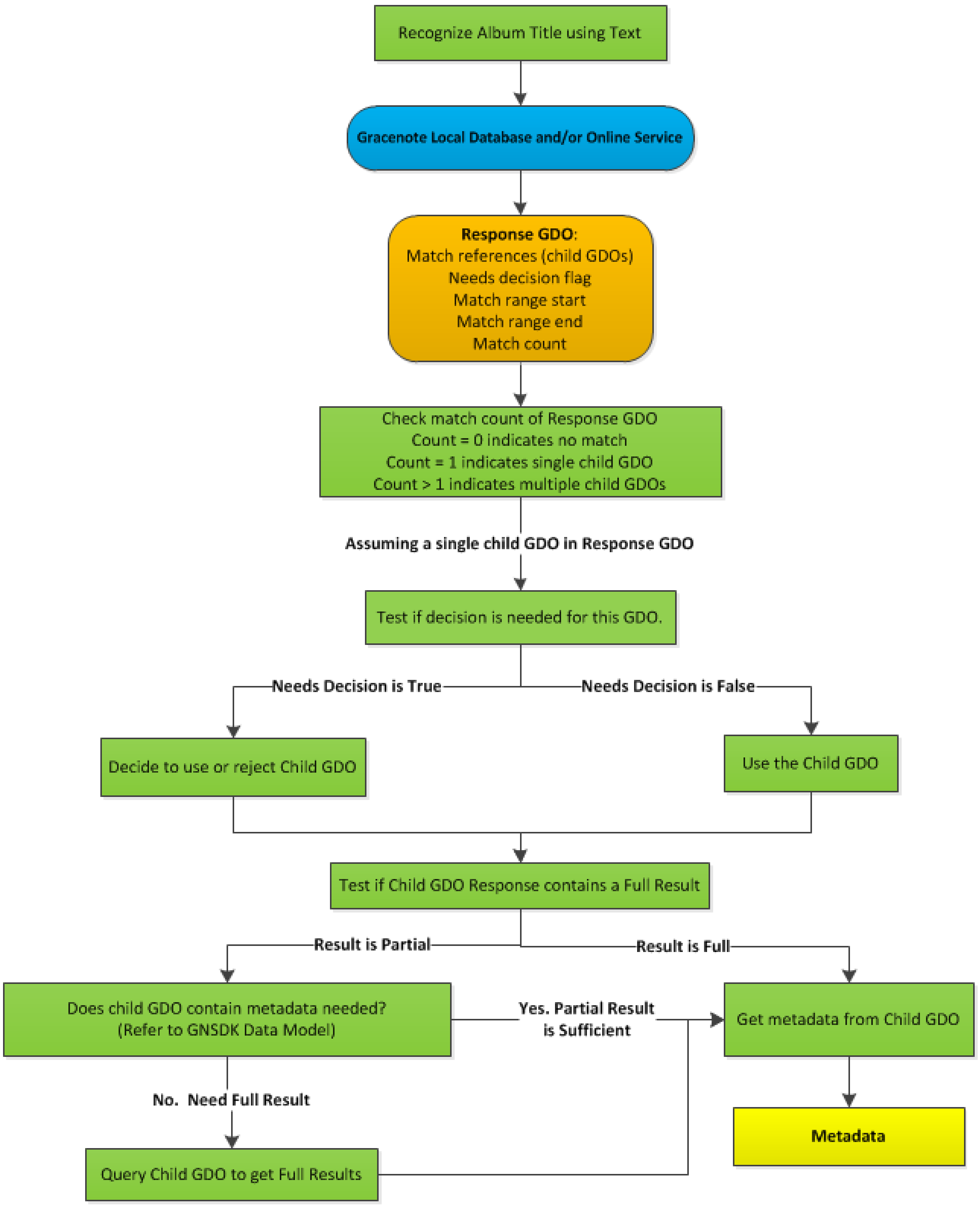
The simplest example of a GDO workflow is when an identification query returns a single match. For example, suppose there is a query to look up a track by name and find its containing album. If that Track only exists on one album, GNSDK for Desktop identifies the single album and returns a response GDO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

As described in "GDO Workflows" on page 85, 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 for Desktop C API Reference*.

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


Recognize Album Title using Text
Gracenote Local Database and/or Online Service
Response GDO:
Match references (child GDOs)
Needs decision flag
Match range start
Match range end
Match count
Check match count of Response GDO
Count = 0 indicates no match
Count = 1 indicates single child GDO
Count > 1 indicates multiple child GDOs
Assuming a single child GDO in Response GDO
Test if decision is needed for this GDO.
Needs Decision is True
Needs Decision is False
Decide to use or reject Child GDO
Use the Child GDO
Test if Child GDO Response contains a Full Result
Result is Partial
Result is Full
Does child GDO contain metadata needed?
(Refer to GNSDK Data Model)
Yes. Partial Result is Sufficient
Get metadata from Child GDO
No. Need Full Result
Metadata
Query Child GDO to get Full Results

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 "GDO Workflows" on page 85

GDO Workflow for Multiple GDO Matches

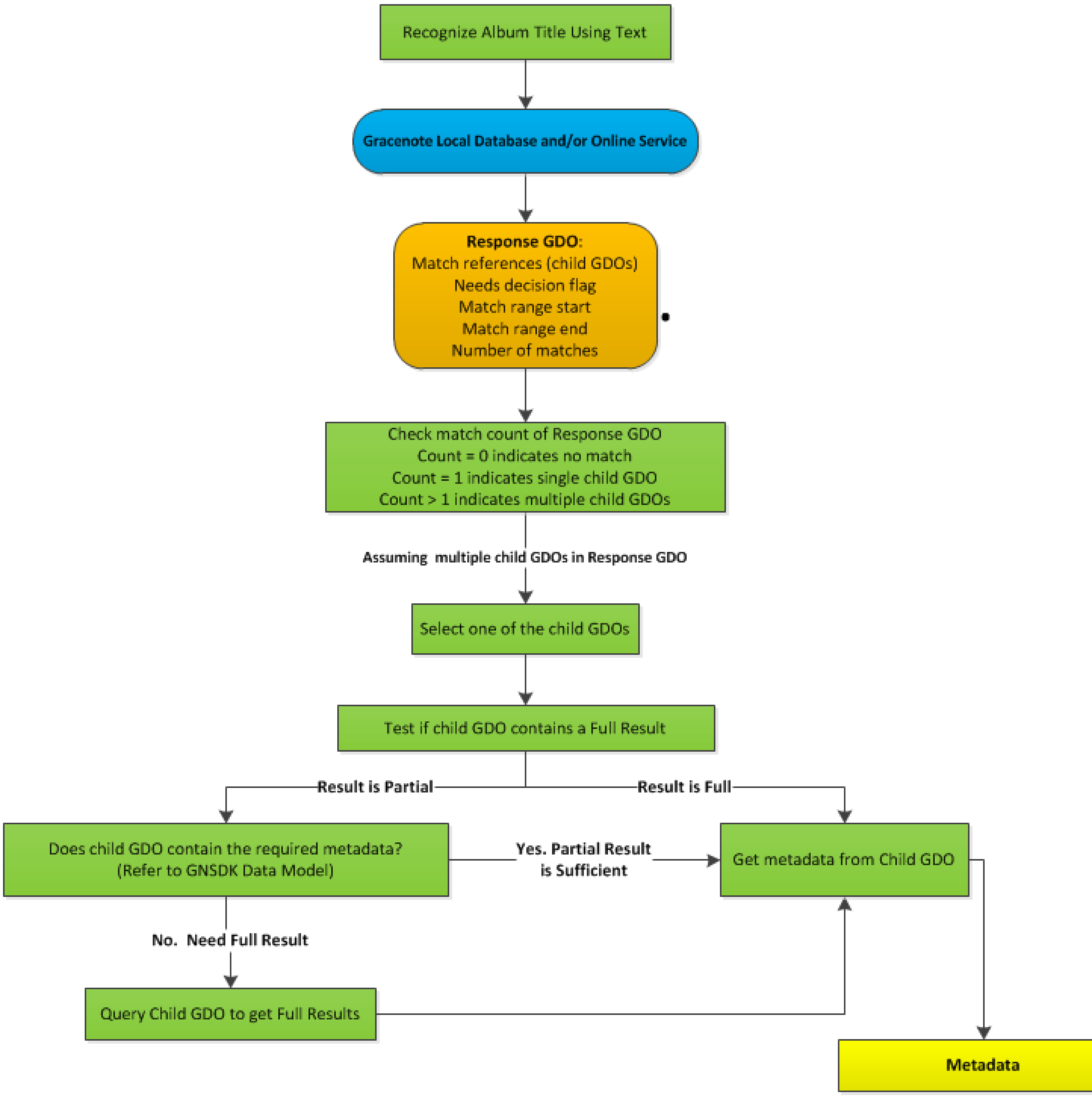
Gracenote identification queries often return multiple matches. For example, suppose that a Track exists on multiple Albums, such as the original Album, a compilation Album, and a greatest hits Album. In this case, the query returns a Response GDO that contains multiple child Album GDOs. Each GDO represents 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 "GDO Workflows" on page 85, *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 in the *GNSDK for Desktop C API Reference*.

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


Recognize Album Title Using Text
Gracenote Local Database and/or Online Service
Response GDO:
Match references (child GDOs)
Needs decision flag
Match range start
Match range end
Number of matches
Check match count of Response GDO
Count = 0 indicates no match
Count = 1 indicates single child GDO
Count > 1 indicates multiple child GDOs
Assuming multiple child GDOs in Response GDO
Select one of the child GDOs
Test if child GDO contains a Full Result
Result is Partial
Result is Full
Does child GDO contain the required metadata? (Refer to GNSDK Data Model)
Yes. Partial Result is Sufficient
Get metadata from Child GDO
No. Need Full Result
Query Child GDO to get Full Results
Metadata

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 "GDO Workflows" on page 85

Related Information

Using Gracenote Data Objects 82

Common GDO Tasks

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.

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 */
```

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_musicid_query_set_gdo(query_handle, album_gdo);

    /* Query for this match in full*/
    gnsdk_musicid_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);

    ...
```

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 (e.g., 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 .

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 (e.g., 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  */

}
```

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 for Desktop 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.

Rendering a GDO as XML

GNSDK for Desktop 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 209.

Related Information

GDO Navigation Examples

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

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¿œèµ°é«˜é£ž
          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
```

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 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 TUI identifier
4. Get album child GDO from album response GDO
5. Access and display album metadata
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)

*****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
```

Related Information

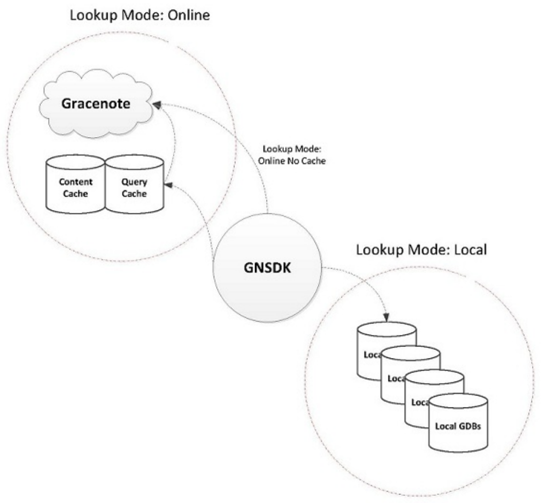
Storage and Caching

Using SQLite for Storage and Caching

The GNSDK for Desktop 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.

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

In the future, other database modules will be made available, but currently, the only option is SQLite. Besides APIs specific to SQLite, there is a set of general storage APIs that apply, now and in the future, to whatever database module is implemented. These general APIs 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_CONTENTCACHE—Stores cover art and related information.
2. GNSDK_MANAGER_STORAGE_QUERYCACHE—Stores media identification requests.
3. GNSDK_MANAGER_STORAGE_LISTSCACHE—Stores Gracenote locale lists.

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);
```

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.

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

```
gnsdk_storage_sqlite_shutdown();
```

Following initialization, your code **must** make the following API call to establish a valid storage folder path for these caches. The path must be writeable. The following sample call sets this folder to the current directory ('.').

```
gnsdk_storage_sqlite_option_set(GNSDK_STORAGE_SQLITE_OPTION_STORAGE_FOLDER, ".");
```

Folder paths can be either relative or absolute. The SDK is path convention agnostic in that, given either direction for slashes (e.g., "\" or "/"), it will correct to the native OS's standard.

In addition to setting the location for all 3 caches, your application has the option to set a location for each cache with the gnsdk_manager_storage_location_set() API. Note that this is a general storage API call and not specific to SQLite.

```
gnsdk_manager_storage_location_set(GNSDK_MANAGER_STORAGE_QUERYCACHE, "./querycache");
```

You might want to 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.

Important: Neither gnsdk_storage_sqlite_option_set() nor gnsdk_manager_storage_location_set() returns an error if the passed location does not exist or is not writeable. In this case, no caching is done. Your application can create this folder at a later time to turn on caching.

Other SQLite API calls cover setting cache file and memory size, journaling, and synchronous operations. See the API Reference for a complete list.

Implementing Status Callbacks

GNSDK for Desktop 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
- MoodGrid presentation handle creation
- Submit parcel handle creation
- Video queries

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

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 for Desktop C API Reference.

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_musicid_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()

Cancelling 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.

Example: Using a Callback

Sample Application: musicid_file_libraryid/main.c

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

Using Locales

GNSDK for Desktop 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 212.

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 Playlist 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, Playlist 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
 );
```

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 for Desktop 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 for Desktop 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.

Locale Groups

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

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, Playlist, etc. 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.

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).

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.

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
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	

Locale/List-Dependent Values	Locale/List Types
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.

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_VIDEOREPUTATION
GNSDK_GDO_VALUE_SCENARIO	GNSDK_LIST_TYPE_VIDEOSCENARIO
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	

Locale/List-Dependent Values	Locale/List Types
GNSDK_GDO_VALUE_VIDEO_PRODUCTION_ TYPE	GNSDK_LIST_TYPE_VIDEOTYPES
GNSDK_GDO_VALUE_VIDEO_REGION	GNSDK_LIST_TYPE_VIDEOREGIONS
GNSDK_GDO_VALUE_VIDEO_REGION_DESC	GNSDK_LIST_TYPE_VIDEOREGIONS
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
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

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

Multi-Threaded Access

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

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

Updating Locales and Lists

GNSDK for Desktop 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 96.

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.

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 retreive 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 78.

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.

Practice	Description
Use multiple threads when loading or updating multiple locales.	Loading locales in multiple threads allows lists to be fetched concurrently, reducing overall load time.
Update locales in a background thread.	Locales can be updated while the application performs normal processing. The SDK automatically switches to using new lists as they are updated. ⚠ If the application is using the GNSDK Manager Lists interface directly and the application holds a list handle, that list is not released from memory and the SDK will continue to use it.
Set a *persistence* flag when updating. If interrupted, repeat update.	If the online update procedure is interrupted (such as network connection/power loss) then it must be repeated to prevent mismatches between locale required lists. Your application should set a persistence flag before starting an update procedure. If the flag is still set upon startup, the application should initiate an update. You should clear the flag after the update has completed.
Call gnsdk_manager_storage_compact() after updating lists or locales.	As records are added and deleted from locale storage, some storage solutions, such as SQLite, can leave empty space in the storage files, artificially bloating them. You can call gnsdk_manager_storage_compact() to remove these. ⚠ The update procedure is not guaranteed to remove an old version of a list from storage immediately because there could still be list element references which must be honored until they are released. Therefore, your application should call gnsdk_manager_storage_compact() during startup or shutdown after an update has finished.

Example: Loading a Locale and Retrieving Locale-Sensistive 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.

Testing an Application

Gracenote helps you validate and ship your Gracenote-enabled application. Read the following sections carefully to learn the procedures you must follow to release your application.

Enabling Test Mode

When you are ready to begin testing your Gracenote-enabled application, send an email to DevSupport@gracenote.com to register your client application. Gracenote support will send back information that will allow you to test your application.

Setting Environment Variables

Some GNSDK for Desktop 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 for Desktop 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.
GNSDK_DEBUG_LOG=<log filename>	Set this variable to a valid file name to enable GNSDK for Desktop debug logging to the given file name.
GNSDK_PROXY_HOST=hostname:port	Set this variable to set GNSDK for Desktop 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.

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.

Implementing Recognition Features

Recognizing Music

Music Recognition Overview

GNSDK for Desktop 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 sourc

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 12.

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 for Desktop 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«˜é£ž
          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
```

Related Information

Identifying Music Using Text

You can identify music using a text string lookup. For more information, see "Text-Based Recognition" on page 13.

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:

1. GNSDK_MUSICID_FIELD_ALBUM (album title)
2. GNSDK_MUSICID_FIELD_TITLE (track title)
3. GNSDK_MUSICID_FIELD_ALBUM_ARTIST (album artist)
4. GNSDK_MUSICID_FIELD_TRACK_ARTIST (track artist)
5. 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 );
```

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:

1. GNSDK_GDO_TYPE_ALBUM
2. 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 143.

For more information about navigating through the GDO type hierarchy, see "About Gracenote Data Objects (GDOs)" on page 82.

Text searches can return multiple matches containing partial results. For information on handling this, see "Full and Partial Metadata Results" on page 83

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    : çœ‹æˆ‘72å
    Match count: 10
          Title: çœ‹æˆ‘72è®Š
          Title: çœ‹æˆ‘72å
          Title: çœ‹æˆ‘72å
          Title: çœ‹æˆ‘72è®Š
          Title: çœ‹æˆ‘72è®Š
          Title: çœ‹æˆ‘72å
          Title: çœ‹æˆ‘72è®Š
          Title: çœ‹æˆ‘72å
          Title: çœ‹æˆ‘72å
          Title: çœ‹æˆ‘72å
    Final match:
          Title: çœ‹æˆ‘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
```

Related Information

Text-Based Recognition..13

Identifying Music Using Fingerprints

You can identify music using an audio fingerprint. For more information, see "Fingerprint-Based Recognition" on page 13.

PCM Audio Format Recommendations

GNSDK for Desktop fingerprinting supports the following PCM Audio Formats:

Sample Sizes:

- 16-bit

Channels:

- 2 or 1 (stereo or mono)

Sample Rates:

- 48000 Hz
- 44100 Hz
- 32000 Hz
- 24000 Hz
- 22050 Hz
- 16000 Hz
- 11025 Hz

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

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

Example: Identifying Music from an Audio Stream

Sample Application: musicid_stream/main.c

Application Steps:

1. Authenticate caller and initialize GNSDK Manager, SQlite, DSP, and MusicID.
2. Enable logging, load a locale and get a User handle.
3. Create query and set with fingerprints.
4. Perform query and display results.
5. Release resources and shutdown GNSDK and modules.

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
```

Related Information

Implementing Radio

Radio:

- Is a subset of MusicID-Stream.
- Recognizes streaming audio from traditional radio sources, including AM/FM, HD, DAB, and others.
- Recognizes music automatically without the need to start or stop recognition.
- Supports broadcast metadata such as title, artist, or station.

Implementing Radio in an application involves the following steps:

1. Initializing the MusicID-Stream module.
2. Registering callbacks
3. Creating a MusicID-Stream channel
4. Listen to the audio source and handling callbacks.
5. Releasing the channel.

Initializing the MusicID-Stream Module

Before using Radio, follow the usual steps to initialize GNSDK for Desktop. The following GNSDK for Desktop modules must be initialized:

- GNSDK Manager
- GNSDK Logging (optional)
- SQLite
- DSP
- Locale (optional)
- User
- MusicID

For more information on initializing the GNSDK for Desktop modules, see “Initializing an Application.”

To initialize Radio, use the gnsdk_musicidstream_initialize() function.

```
error = gnsdk_musicidstream_initialize(sdkmgr_handle);
```

Registering Callbacks

Radio listens to an audio stream and identifies music without the need for user intervention. Radio returns information to the application through the use of callbacks. The following callbacks are available:

- **`callback_status`**: Used during identification when an online query is performed.
- **`callback_processing_status`**: Provide this callback only if Gracenote Global Services and Support advises it.
- **`callback_identifying_status`**: Used during audio processing when a significant event occurs such as large classification change or transition.
- **`callback_result_available`**: Called when a result has been determined by MusicID-Stream for a given audio stream within a channel.
- **`callback_error`**: Called if an error occurs during identification.

It is usually not necessary to use `callback_status` or `callback_processing_status` for Radio.

Creating a MusicID-Stream Channel

The callback functions must be registered with MusicID-Stream when you create a channel. The channel is how MusicID-Stream listens to the audio. The following code sets up the callbacks for use with the channel, and provides a channel handle. It uses the channel creation option `gnsdk_musicidstream_preset_radio`, which indicates that the audio is coming in from a direct line in.

```
gnsdk_musicidstream_callbacks_t callbacks      = {0};

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,          /* User callback functions */
       GNSDK_NULL,          /* Optional data passed to the callbacks */
       &channel_handle
);
```

Once you have a channel handle, you enable automatic recognition mode using `gnsdk_musicidstream_channel_automatic_set()`.

You can create a .wav file to fingerprint your audio using `GNWaveFileCreate()`. Once you have a .wav file you can fingerprint it using `gnsdk_musicidstream_channel_audio_begin()`, and stop listening with `gnsdk_musicidstream_channel_audio_end()`.

Handling Callbacks

Each callback has its own signature and passes information to your application as it is generated. When results from an identification occur, the `callback_result_available` is called. It is passed in a response GDO containing the identifying information of the album on the channel. This GDO can be used to display information, or for further processing as input into other Gracenote software.

Shutting Down Radio

When you are finished using Radio, release the channel using `gnsdk_musicidstream_channel_release()`. After the channel is released, shutdown the GNSDK for Desktop modules you are using in the reverse order in which they were initialized.

Recognizing Video

Using VideoID

Besides acquiring the necessary licensing, including VideoID requires the following define:

```
#define GNSDK_VIDEO 1
```

And making the following initializing API call with your SDK Manager handle:

```
gnsdk_video_initialize(sdkmgr_handle);
```

Before your program exits, it needs to make the following API to release VideoID resources:

```
gnsdk_video_shutdown();
```

Recognizing Video Products with VideoID

GNSDK provides two modules for implementing video features: VideoID and VideoExplore:

1. **VideoID** supports recognizing **Products** and enables access to related metadata such as title, genre, rating, synopsis, cast and crew. 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 (Univeral Product Code), Hinban, or EAN (European Article Number). Products are for the most part released on a physical format, such as a DVD or Blu-ray.
2. **Video Explore** enables advanced recognition to video elements and their metadata. You can use VideoExplore to navigate among Products, AV Works, Contributors, Series, and Seasons. For more information, "VideoExplore Overview"

You can use the following video identifiers in VideoID recognition queries:

- TOC (table of contents)
- Text - You can search on product title, character name, contributor name, series title, Work franchise, Work series, and Work title
- External ID, such as UPC, EAN, and Hinban codes
- GDO other Gracenote ID for the Product, such as a GNID, or TUI/TUI Tag combination.

The following code samples use Text and a UPC in a recognition query. For an example using a TOC, "Find Video Products by TOC".

Text

```
/* Create query handle */
gnsdk_video_query_create(user_handle, _video_status_callback_fn, GNSDK_NULL, &query_handle);

/* Set the search text */
gnsdk_video_query_set_text(
      query_handle,
      GNSDK_VIDEO_SEARCH_FIELD_PRODUCT_TITLE, // Search on product title
      "spider",                              // Search text
      gnsdk_video_search_type_anchored       // Results must begin with specified text
);

/* Query for suggestions based on text */
gnsdk_video_query_find_suggestions(query_handle, &response_gdo);
```

UPC code

```
/* Create query handle */
gnsdk_video_query_create(user_handle, GNSDK_NULL, GNSDK_NULL, &query_handle);

/* Set the input ID */
gnsdk_video_query_set_external_id(query_handle,
      "025192032721",
      GNSDK_VIDEO_EXTERNAL_ID_TYPE_UPC,
      GNSDK_VIDEO_EXTERNAL_ID_SOURCE_DEFAULT);

/* Attempt to look up this UPC */
gnsdk_video_query_find_products(query_handle, &response_gdo);
```

Related Information

VideoID Overview.. 18

Find Video Products by TOC

Below is sample code showing a find query using a Product's TOC.

```
/* Create query handle */
gnsdk_video_query_create(user_handle, GNSDK_NULL, GNSDK_NULL, &query_handle);

/* Set the input ID */
gnsdk_video_query_set_toc_string( query_handle,
"1:15;2:198 15;3:830 7241 6099 3596 9790 3605 2905 2060 10890 3026 6600 2214
5825 6741 3126 6914 1090 2490 3492 6515 6740 4006 6435 3690 1891 2244 5881 1435 7975 4020
4522 2179 3370 2111 7630 2564 8910 15;4:830 7241 6099 3596 9790 3605 2905 2060 10890 3026
6600 2214 5825 6741 3126 6914 1090 2490 3492 6515 6740 4006 6435 3690 1891 2244 5881 1435
7975 4020 4522 2179 3370 2111 7630 2564 8910 15;5:8962 15;6:11474 15;7:11538 15;", 0 );

/* Attempt to look up this TOC */
gnsdk_video_query_find_products(query_handle, &response_gdo);
```

The example below demonstrates using VideoID to perform a Products query (gnsdk_video_query_find_products()). While this example focuses on looking up a video Product using a TOC, the basic query steps are similar for most GNSDK lookups.

Sample code:

video_products_lookup/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle.
2. Create query using TOC.
3. Get Product GDO.
4. Get Product title GDO and display title.
5. Access and display Product metadata
6. Release resources and shutdown GNSDK and Video module.

When you run this program, you should see the following output, though the GNSDK versions are likely to be different.

Sample output:

```
GNSDK Product Version   : 3.2.0.274     (built 2012-11-30 09:09-0800)

Title: A Bug's Life
Aspect ratio: 2.35:1
Production Type: Motion Picture
Package language: English
Rating: G
```

Implementing Advanced Recognition and Organization Features

Using the MusicID-File APIs

In general, your application should take the following steps when using one of the MusicID-File methods for identification - TrackID, AlbumID or LibraryID:

TrackID and **AlbumID**

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 submit and add 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. Handle query results and statuses
8. Release resources and shutdown MusicID-File, DSP, and GNSDK

LibraryID

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, etc.
3. Create a MusicID-File query handle with callbacks array
4. Create a FileInfo object for each media file you want to submit and add to 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, handle query results and statuses
9. Release resources and shutdown MusicID-File, DSP, and GNSDK

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 */
```

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 calback */
      _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 */
   );
```

Creating FileInfo Objects

For each media file that you want to submit, you need to create a file information object for it (this adds it to the query handle at the same time) and get its handle:

```
/*
 * Create a File Info object and add to query
 */
gnsdk_musicidfile_query_fileinfo_create(
      query_handle,              /* MusicID-File query handle to create FileInfo for */
      "data/01_stone_roses.wav", /* Media file path */
      GNSDK_NULL,                /* No callback function for status and progress */
      GNSDK_NULL,                /* No callback data */
      &fileinfo_handle           /* Pointer to receive created FileInfo */
    );
```

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

Note: FileInfo objects are freed when the MusicID-File query handle is released.

Setting MusicID-File Query Options

To set an option for your MusicID-File query, use the gnsdk_musicidfile_query_option_set() API. 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.

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

Besides local lookup, there are options for returning the following: alternate names for contributor data, classical music data, fetching content (e.g., images) data, external IDs, playlist data, and sonic 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.

Setting Media File Identification

For TrackID and AlbumID, setting media file identification is done before making the query, for LibraryID, this is done after the query is made through callbacks.

Fingerprinting

The MusicID-File fingerprinting APIs give your application the ability to provide audio data as an identification mechanism. This enables MusicID-File to perform identification based on the audio itself, as opposed to performing identification using only the associated metadata. Use the MusicID-File fingerprinting APIs either before processing has begun (TrackID or AlbumID), or during a gnsdk_ musicidfile_callback_get_fingerprint_fn callback (LibraryID).

There are 3 fingerprinting APIs:

- **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 to generate the fingerprint for */
      11025,             /* Sample frequency of audio to be provided: 11 kHz, 22 kHz, or 44 kHz */
      16,                /* Sample rate of audio to be provided (in 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 to generate the fingerprint for */
      pcm_audio,         /* Pointer to audio data buffer that matches the 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 Metadata

Use the gnsdk_musicidfile_fileinfo_metadata_set() and gnsdk_musicidfile_fileinfo_metadata_get() APIs to set and get metadata information for a FileInfo object. Note that MusicID-File will not process FileInfos that do not contain metadata.

For 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"
   );
```

Other metadata options include getting/setting CDDB IDs, fingerprint, path and filename, FileInfo identifier (set when FileInfo created), media ID (from Gracenote), source filename (from parsing) and application, Media Unique ID (MUI), Tag ID (aka Product ID), TOC offsets, track artist/number/title, and TUI (Title Unique Identifier). See the API reference for a complete list and more information.

Performing the Query

Each ID method 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.

Example query:

```
/*
 * 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);
```

Note that you can get a query's status at any time with the gnsdk_musicidfile_query_status() API

Getting Results

After making your query, 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 milliseconds or until done (INFINITE)
*/
      GNSDK_NULL                            /* Pointer to error returned upon completion */
   );
```

TrackID and **AlbumID**

Once the query completes, you can 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 each file in the query...
 */
for (i = 0; i < count; i++)
{
    gnsdk_musicidfile_query_fileinfo_get_by_index(query_handle, i, &fileinfo_handle);
    ...
}
```

Note that you also have the option to assign an identifer string when creating a FileInfo object and then use the identifier to retrive the FileInfo object with the gnsdk_musicidfile_query_fileinfo_get_by_ident() API

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_musicidfile_fileinfo_status_t  fileinfo_status = gnsdk_musicidfile_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_musicidfile_fileinfo_status(fileinfo, &fileinfo_status, &p_error_info);
if (gnsdk_musicidfile_fileinfo_error != fileinfo_status)
{
    if ((gnsdk_musicidfile_fileinfo_result_single == fileinfo_status) ||
        (gnsdk_musicidfile_fileinfo_result_all == fileinfo_status))
    {
        gnsdk_musicidfile_fileinfo_get_response_gdo(fileinfo, &response_gdo);

        ...Parse Response GDO
    }
}
```

LibraryID

With LibraryID, results are returned in a results available callback:

```
static gnsdk_void_t    GNSDK_CALLBACK_API
       _result_available_callback(
            const gnsdk_void_t*               user_data,
            gnsdk_musicidfile_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() */
```

In LibraryID, note that no additional call needs to be made to get the Response GDO, it is simply passed to the callback.

For all ID methods, once you have the Response GDO, you can access and display GDO metadata as usual.

Releasing Resources and Shutting Down

In addition to the usual releasing of resources (user handle, locale handle, GDOs, etc.), the following are specific to MusicID-File:

```
/*
 * Release MusicID-File - this releases all associated FileInfo objects
 */
gnsdk_musicidfile_query_release(query_handle);
gnsdk_musicidfile_shutdown();
```

Note the following:

- 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.

Related Information

MusicID-File Overview...14

Performing Advanced Music Identification

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‚‰å¯ ã‚‹ã ¾ã §è‹±ä¼šè©±ã ¾ã‚‹ã ”ã ¨ç·´ç¿’å¸³2
       Match 5 - Album title:          ç¬¬66å›ž ä¸å›½èªžæ¤œå®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·ã  ã ¦ã ‹ã‚‰å¯ ã‚‹ã ¾ã §è‹±ä¼šè©±ã ¾ã‚‹ã ”ã ¨ç·´ç¿’å¸³2
       Match 5 - Album title:          ç¬¬66å›ž ä¸å›½èªžæ¤œå®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:          SÃ¥nger Om KÃ¤rlek [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 resouces 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
```

Related Information

MusicID-File Overview..14

Performing Advanced Video Recognition (Video Explore)

VideoExplore Find Queries

This section lists commonly-used Video APIs. Refer to the GNSDK API Reference for a complete list of Video APIs and their usage.

The most commonly used APIs are the gnsdk_video_query_find_*() APIs used to access AV Works, Contributors, Series, Seasons, Programs, and Suggestions metadata:

- gnsdk_video_query_find_contributors()
- gnsdk_video_query_find_seasons()
- gnsdk_video_query_find_series()
- gnsdk_video_query_find_works()
- gnsdk_video_query_find_objects()
- gnsdk_video_query_find_products()
- gnsdk_video_query_find_programs()
- gnsdk_video_query_find_suggestions()

You can use the gnsdk_video_query_find_objects() function to access the GDO associated with a particular input GDO, or all the GDOs associated with an External ID used as input.

GDO Inputs for Video Find Queries

This topic lists the valid Partial and Full GDO inputs for gnsdk_video_query_find*().

Valid Partial GDO Inputs:

Video Find Function	Partial Products	Partial AV Works	Partial Contributors	Partial Seasons	Partial Series
gnsdk_video_query_find_product()	Yes	Yes			

gnsdk_video_query_find_work()	Yes	Yes	Usually identical to input GDO**	Yes	Yes
gnsdk_video_query_find_contributor()		Yes	Yes	Yes	Yes
gnsdk_video_query_find_season()		Yes	Yes	Yes	Yes
gnsdk_video_query_find_series()		Yes	Yes	Yes	Yes

Valid Full GDO Inputs:

Video Find Function	**Full Products**	**Full AV Works**	**Full Contributors**	**Full Seasons**	**Full Series**
gnsdk_video_query_find_product()	Usually identical to input GDO**	Yes			
gnsdk_video_query_find_work()	Yes	Yes	Yes	Yes	Yes
gnsdk_video_query_find_contributor()		Yes	Usually identical to input GDO**	Yes	Yes
gnsdk_video_query_find_season()		Yes	Yes	Usually identical to input GDO**	Yes
gnsdk_video_query_find_series()		Yes	Yes	Yes	Usually identical to input GDO**

**Calling a Full GDO from its corresponding gnsdk_video_query_find_*() function usually returns a GDO that is identical to the input GDO.

Related Information

VideoExplore Overview..19

Accessing Video GDOs from other Video GDOs

The following table shows a subset of the data model for video GDOs and which GDOs can access other GDOs. For a complete list of GDOs and their relationships, see the Data Model in the *GNSDK for Desktop*

C API Reference.

Input GDOs	GDOs Accessible from Input GDOs
Products	• AV Works
AV Works	• Contributors • Products • Seasons • Series AV Work GDOs has the following two optional relationships: • Series: Certain AV Work are standalone and are not categorized by Series. • Seasons: Some Series are not categorized by Seasons.
Contributors	• Seasons • Series • AV Works
Seasons	• Contributors • Series • AV Works
Series	• Contributors • Seasons • AV Works Series GDOs can access two types of AV Work GDOs: 1. AV Works associated with (organized by) Seasons 2. AV Works not associated with Seasons GDOs, such as television pilots and specials, and any television series not categorized by Seasons.

Find Works

This example demonstrates using VideoExplore to perform a Works query (gnsdk_video_query_find_works ()). While this example focuses on finding Works using a contributor name (Harrison Ford), the basic query steps are similar for most GNSDK lookups.

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.

Sample code:

video_works_lookup/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle.
2. Create query based on contributor text search.
3. Perform find works query.
4. Loop through child Work GDOs returned, accessing and displaying titles.
5. Release resources and shutdown GNSDK and Video module.

When you run this program, you should see the following output, though the GNSDK versions you see are likely to be more recent.

Sample output:

```
GNSDK Product Version  : 3.2.0.274     (built 2012-11-30 09:09-0800)

AV Works for Harrison Ford:
       1: Cowboys & Aliens
       2: Star Wars: Episode VI - Return Of The Jedi
       3: Star Wars: Episode IV - A New Hope
       4: Sabrina
       5: Indiana Jones And The Kingdom Of The Crystal Skull
       6: Six Days, Seven Nights
       7: American Graffiti
       8: The Fugitive
       9: Indiana Jones And The Last Crusade
       10: Apocalypse Now
```

Find Seasons

This example demonstrates using VideoExplore to perform a Seasons query (gnsdk_video_query_find_seasons()). While this example focuses on looking up a video Season using a TUI and TUI tag, the basic query steps are similar for most GNSDK lookups.

A Season is 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).

Sample code:

video_seasons_lookup/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, load a locale, enable SDK logging, and get a user handle
2. Create query from TUI and TUI Tag
3. Get Seasons GDO
4. Display title and other metadata
5. Release resources and shutdown GNSDK and Video module

Sample output:

```
GNSDK Product Version  : 3.2.0.274     (built 2012-11-30 09:09-0800)

*****Sample Video Seasons Search: 'Simpsons First Season'*****

Perform the search...

Title: Season 1

Episodes:
       1 : Simpsons Roasting on an Open Fire
       2 : Bart the Genius
       3 : Homer's Odyssey
       4 : There's No Disgrace Like Home
       5 : Bart The General
       6 : Moaning Lisa
       7 : The Call of the Simpsons
       8 : The Telltale Head
       9 : Life on the Fast Lane
       10 : Homer's Night Out
       11 : The Crepes of Wrath
       12 : Krusty Gets Busted
       13 : Some Enchanted Evening
```

Find Series

This example demonstrates using VideoExplore to perform a basic Series query (gnsdk_video_query_find_series()). While this example focuses on looking up a Video Series using a TUI and TUI tag, the basic query steps are similar for most GNSDK lookups.

A Series is a collection of related Works, typically in sequence, and often comprised of Seasons (generally for a TV series), for example: CSI: Miami, CSI: Vegas, CSI: Crime Scene Investigation.

Sample code:

video_series_lookup/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle
2. Create query from TUI and Tui Tag
3. Get Series GDO
4. Get and display title and other metadata
5. Release resources and shutdown GNSDK and Video module

Sample output:

```
GNSDK Product Version   : 3.2.0.274     (built 2012-11-30 09:09-0800)

*****Sample Video Series Search: 'Simpsons Series'*****

Perform the search...
```

```
Title: The Simpsons
Genre: Animation
Production Type: TV Series
Plot: Homer and Marge Simpson raise Bart, Lisa and baby Maggie.
Reputation: Critical Darling
```

Find Contributors

This example demonstrates using VideoExplore to perform a Contributors query (gnsdk_video_query_find_contributors()). While this example looks up a video work using a deserialized GDO, the basic query steps are similar for most GNSDK lookups.

A Contributor refers to any person who plays a role in a Work. Actors, Directors, Producers, Narrators, and Crew are all considered Contributors. Popular recurring Characters such as Batman, Harry Potter, or Spider-man are also considered Contributors.

Sample code:

video_contributors_lookup/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle
2. Create query from deserialized GDO
3. Get AV Work GDO
4. Get and display Work title
5. Get AV Work Contributors and display
6. Release resources and shutdown GNSDK and Video module

When you run this program, you should see the following output, though the GNSDK versions are likely to be more recent.

Sample output:

```
GNSDK Product Version  : 3.2.0.274     (built 2012-11-30 09:09-0800)

Title: The Dark Knight

Contributors:
       1 : Christian Bale
       2 : Heath Ledger
       3 : Aaron Eckhart
       4 : Michael Caine
       5 : Maggie Gyllenhaal
       6 : Gary Oldman
       7 : Morgan Freeman
       8 : Monique Gabriela Curnen
       9 : Ron Dean
       10 : Cillian Murphy
```

Find Suggestions

This example demonstrates using VideoExplore to perform a Suggestions query (gnsdk_video_query_find_suggestions()). While this example searches for Products based on title, the basic query steps are similar for most GNSDK lookups.

Possible items your application can search on include Character name, Contributor name, Product title, Series title, Work franchise, Work series and Work title.

Your license determines what suggestion searches your app can perform:

- To query for Product titles requires VideoID or Video Explore licensing
- To query for Works titles, Series titles, or Contributor names requires Video Explore licensing.

Sample Code:

video_search_suggestion/main.c

Application steps:

1. Authenticate caller, initialize GNSDK Manager and Video module, enable SDK logging, load a locale, and get a user handle.
2. Create a video query.
3. Set the query handle with range options and search term ("spider").
4. Perform a find suggestions query.
5. Loop through each suggestion returned, displaying results.
6. Release resources and shutdown GNSDK and Video module.

When you run the program, you should see the following output, though your GNSDK versions are likely to be more recent.

Sample output

```
GNSDK Product Version   : 3.2.0.274     (built 2012-11-30 09:09-0800)

Info: No stored user - this must be the app's first run.

*****Sample VideoID Suggestion Lookup*****

.........     1 - 20 of 56 suggestions for 'spider'
       1: Spider-Man 1
       2: Spider-Man 2
       3: Spider-Man 3
       4: Spider-Man
       5: Spider-Man Trilogy: Spider-Man / Spider-Man 2 / Spider-Man 3
       6: Spider-Man: The '67 Classic Collection
       7: Spider-Man: The High Definition Trilogy
       8: spider-Man - The Venom Saga
       9: Spider-Man: The Ultimate Villain Showdown
       10: Spider-Man Vs. Doc Ock
       11: The Spiderwick Chronicles
       12: Spider Riders #1
       13: Spider-Man: The New Animated Series
       14: Spider-Man: La TrilogÃa
       15: Spider's Web
       16: Spider-Man: Die High Definition Trilogie
       17: Spider Baby
       18: Spider-Man: The Return Of The Green Goblin
       19: Spider
       20: Spiders
....   21 - 40 of 56 suggestions for 'spider'
       21: The Spiders
       22: Spider Forest
       23: Spider
       24: Spiders
       25: Spider-Man: Mutant Agenda
       26: Spider-Man: The New Animated Series - The Complete First Season
       27: Spider-Man: The New Animated Series #2 - High-Voltage Villains
       28: Spider-Man 3-Pack: Daredevil Vs. Spider-Man / Return Of The Green Goblin / Ultimate Villain
Showdown
       29: Spider-Man: La Trilogia
       30: Spider Murphy Gang: 30 Jahre Rock 'N' Roll
       31: Spiders 2: Breeding Ground
       32: Spider's Web: A Pig's Tale / Plan Bee
       33: Spiderwick Chonicles
       34: Spiderwick: Le Cronache
       35: Spider-Man: Complete - Season 2
       36: Spider-Man DVD-Trilogie
       37: Spider-Man: Trilogie Haute DÃ©finition
       38: Spider-Man: Komplette Staffel 1
       39: Spider-Man: Complete Season 1
       40: SpiderBabe
....   41 - 56 of 16 suggestions for 'spider'
       41: Spider-Man 5000, Vol. 1
       42: Spider-Man: Complete - Season 4
       43: Spider-Man: Complete Season 3
       44: Spider
       45: The Spiders From Mars And Glamorous Angel
       46: Spider-Man: DVD Blast!
       47: Spider-Man: Extreme Threat
       48: Spider-Man: The Mutant Menace
       49: Spider-Man: New Animated Series - High Voltage Villians
       50: Spiders Tale / Plan Bee
       51: Spider: La Vida De Mi Porbenir
       52: The Spider Woman
       53: Spider-Woman: Agent Of S.W.O.R.D.
       54: Spiderbabe: Rated Collector Edition
       55: Spiderbabe: Unrated Collector Edition
       56: Spiderbait: Greatest Hits
```

Accessing Core Metadata

Accessing Music Metadata

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 82.

> 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 83.

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 accessses 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.

Accessing Mood and Tempo Metadata

In GNSDK for Desktop, access sonic 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 Global Services & Support representative for more information.

Mood

GNSDK for Desktop 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);
```

Tempo

NOTE: Tempo metadata is available online-only.

GNSDK for Desktop 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

Related Information

Mood and Tempo (Sonic Attributes) 8

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.

⚠ Your application must be licensed to access this metadata. Contact Gracenote Global Services & Support for more information.

For more information, see "Classical Music Metadata" on page 9.

The following table lists the applicable query functions and options for the GNSDK for Desktop 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

For an example of the metadata returned with a classical album, see "Classical Album Metadata Example" on page 148.

Related Information

Classical Music Metadata 9

Accessing External IDs

GNSDK for Desktop 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	gnsdk_*_query_option_set()	GNSDK_*_OPTION_ENABLE_EXTERNAL_ID_DATA 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);
        }
    }
}
```

Related Information

Third-Party Identifiers and Preferred Partners...10

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 216.

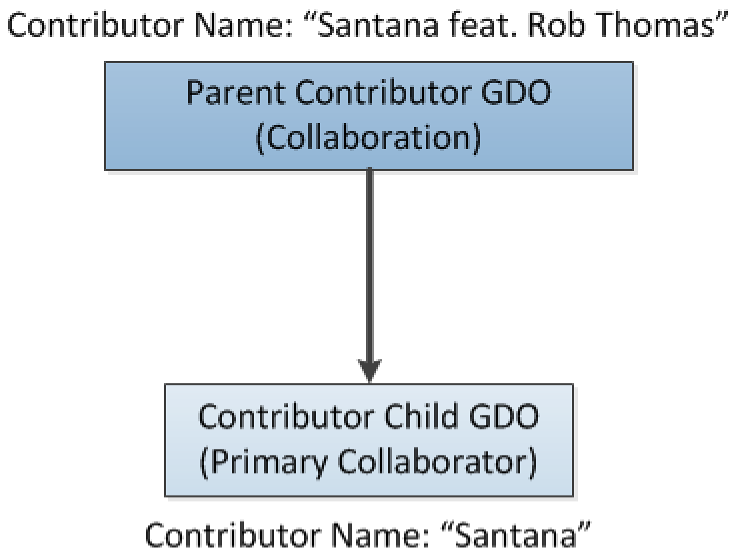
Navigating Collaborations

In GNSDK for Desktop, 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”
Parent Contributor GDO (Collaboration)
Contributor Child GDO (Primary Collaborator)
Contributor Name: “Santana”

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 for Desktop 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 143.
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.

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."

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.

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 109.

Related Information

Music Metadata Reference

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 for Desktop 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 &amp; La La</DISPLAY>
    </NAME_OFFICIAL>
    <CONTRIBUTOR>
      <NAME_OFFICIAL>
        <DISPLAY>Nelly Feat. Cedric The Entertainer &amp; 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 &amp; La La</DISPLAY>
      </NAME_OFFICIAL>
      <CONTRIBUTOR>
        <NAME_OFFICIAL>
          <DISPLAY>Nelly Feat. Cedric The Entertainer &amp; 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>
```

Classical Album Metadata Example

The following is intended to give the reader some idea of the metadata returned in a GDO for a classical album.

```
<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 &amp; 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>JÃ¡nos SÃ¡ndor: Budapest Philharmonic Orchestra</DISPLAY>
      </NAME_OFFICIAL>
      <CONTRIBUTOR>
        <NAME_OFFICIAL>
          <DISPLAY>JÃ¡nos SÃ¡ndor: 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&apos;s</ERA_LEVEL1>
        <ERA_LEVEL2>2000&apos;s</ERA_LEVEL2>
        <ERA_LEVEL3>2000&apos;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&apos;s</ERA_LEVEL1>
        <ERA_LEVEL2>1990&apos;s</ERA_LEVEL2>
        <ERA_LEVEL3>1990&apos;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 Wohlert: Berlin Radio Symphony Orchestra</DISPLAY>
      </NAME_OFFICIAL>
      <CONTRIBUTOR>
        <NAME_OFFICIAL>
          <DISPLAY>Daniel Gerard; Peter Wohlert: 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>
```

Accessing Video Metadata

Getting Video Metadata

Implementing Text Filters

GNSDK for Desktop Video gives your application the ability to filter results based on a number of keys (e.g., genre, production type, etc.) and values (e.g., comedies, drama, documentaries, etc.). Note that the system disregards filters when performing non-text related video queries, and that list element filtering is restricted to performing a Works search.

Sample Code:

video_search_filters/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, load a locale, and get a user handle
2. Perform a contributor AV Work query excluding comedies from search
3. Loop thru AV Works returned, displaying title and genre
4. Release resources and shutdown GNSDK and Video module

When you run the sample, you should see the following output, though your GNSDK for Desktop versions are likely to be different.

Sample output:

```
GNSDK Product Version   : 3.2.0.274     (built 2012-11-30 09:09-0800)

*****Sample Video Work Search: 'Harrison Ford'*****

Number matches: 10

Title: Cowboys & Aliens
Genre: Action/Adventure

Title: Star Wars: Episode VI - Return Of The Jedi
Genre: Sci-Fi & Fantasy

Title: Star Wars: Episode IV - A New Hope
Genre: Sci-Fi & Fantasy

Title: Indiana Jones And The Kingdom Of The Crystal Skull
Genre: Action/Adventure

Title: The Fugitive
Genre: Mystery & Suspense

Title: Indiana Jones And The Last Crusade
Genre: Action/Adventure

Title: Apocalypse Now
Genre: Military & War

Title: Hearts of Darkness: A Filmmaker's Apocalypse
Genre: Documentary

Title: What Lies Beneath
Genre: Mystery & Suspense

Title: Air Force One
Genre: Mystery & Suspense
```

Accessing Locale-Dependent Data

The GNSDK for Desktop provides *locales* as a convenient way to group metadata specific to a region (e.g., Europe) and language (e.g., French) that should be returned from the Gracenote service. What defines a locale is a group (such as Video), 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.

Locale-dependent fields for Video include genre, mood, setting type, origin, and more. For a complete list of Video locale-dependent fields for Video see Locales.htm.

Using locales requires making an API call to load a locale and set it as the video group's default:

```
gnsdk_locale_handle_t locale_handle    = GNSDK_NULL;

/* Setting the 'locale' to return locale-specifc values in English */
gnsdk_manager_locale_load(
       GNSDK_LOCALE_GROUP_VIDEO,     // Group - Video (others include EPG, Playlist and Music)
       GNSDK_LANG_ENGLISH,           // Language - English
       GNSDK_REGION_DEFAULT,         // Default is US (others include China, Japan, Europe)
       GNSDK_DESCRIPTOR_SIMPLIFIED,  // Default music descriptor is 'DETAILED'
       user_handle,                  // User handle
       _list_status_callback_fn,     // Status callback function for loading locale lists
       0,                            // Callback user data
       &locale_handle);              // Locale handle

/* Setting the 'locale' as default
 * If default not set, no locale-specific results would be available
 */
gnsdk_manager_locale_set_group_default(locale_handle);
```

The load operation blocks the current thread until complete. Your app can receive status callback messages via an optional callback function.

Note that loading a locale requires releasing the locale handle on program exit:

```
gnsdk_manager_locale_release(locale_handle);
```

Sample code:

video_locale/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle
2. Create query using deserialized GDO
3. Perform find Works query
4. Access and display locale-dependent metadata from first Work
5. Release resources and shutdown GNSDK and Video module

Sample output:

```
GNSDK Product Version  : 3.2.0.274     (built 2012-11-30 09:09-0800)

*****Sample Video Work Search*****

Number matches: 1
```

```
Title: Crouching Tiger, Hidden Dragon

Origin1: Asia
Origin2: China
Origin3: China
Origin4: China
Genre1: Action/Adventure
Genre2: Adventure
Genre3: Adventure
Rating: PG-13 [MPAA (USA)] Reason: Some Sexuality, Martial Arts Violence

Plot synopsis: Yu Shu Lien (Michelle Yeoh) has been charged with bringing the Green Destiny, a
legendary antique sword, to its new owner in the capital. But the sword is stolen by a mysterious
black-clad thief whose martial arts expertise outshines even that of Shu Lien herself. Li Mu Bai
(Chow Yun Fat) soon arrives to help search for the sword, and his suspicions are aroused when he
realises that his old enemy Jade Fox (Cheng Pei-Pei) is nearby. Ang Lee's epic tale of martial arts
chivalry broke box office records for foreign language film distribution, received critical acclaim
the world over, and won four Oscars (Best Art Direction, Best Cinematography, Best Music, and Best
Foreign Language Film).
```

Accessing AV Work Credits and Filmography

Video Explore helps you expand application capabilities with support for searching among Contributors, Products, Seasons, Series, and Works. It facilitates user exploration and discovery with related queries such as:

- Who are the cast and crew?
- What other films did this actor play in?

Sample Code:

video_explore/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle
2. Perform an AV Work query
3. Get and display all AV Work actor credits
4. Get and display filmography of 1st actor from credit list
5. Release resources and shutdown GNSDK and Video module

When you run the sample, you should see the following output, though your GNSDK versions are likely to be more recent.

Sample output:

```
GNSDK Product Version  : 3.2.0.274     (built 2012-11-30 09:09-0800)

Video Work - Crouching Tiger, Hidden Dragon:
```

```
Actor Credits:
       1 : Michelle Yeoh
       2 : Chang Chen
       3 : Ziyi Zhang
       4 : Pei-pei Cheng
       5 : Lung Sihung
       6 : Fa-zeng Li
       7 : Fa Zeng Li
       8 : Xian Gao
       9 : Yan Hai
       10 : Deming Wang
       11 : Li Li
       12 : Chow Yun Fat

Actor Credit: Michelle Yeoh Filmography

Number works: 10
       1 : The Lady
       2 : Kung Fu Panda 2
       3 : True Legend
       4 : Reign of Assasins
       5 : Babylon A.D.
       6 : The Children Of Huang Shi
       7 : August 30, 2008
       8 : The Mummy: Tomb Of The Dragon Emperor
       9 : July 30, 2008
       10 : Far North
```

Accessing Video Product Metadata

This example demonstrates accessing and parsing metadata from a video Product: Disc > Side > Layer > Feature > Chapter (see sample output below). 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.

Sample Code:

video_product_metadata/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, load a locale, enable SDK logging, and get a user handle
2. Perform title search
3. Select first match and parse/display metadata.
4. Release resources and shutdown GNSDK and Video module

Sample output

```
GNSDK Product Version   : 3.2.0.274     (built 2012-11-30 09:09-0800)

*****Sample Title Search: 'Star'*****
```

```
Possible Matches: 20

Selecting first one:

Title: Star!
Production type: Motion Picture
Orig release: 1968
Rating: G [MPAA (USA)]
Release: 2004-04-20
Discs: 1

Discs:1
disc 1 --------
       Title: Star! Side A: Feature
       Number sides: 2
       Side 1 --------
       Number layers: 1
               Layer 1 --------
               Media type: DVD
               TV system: NTSC
               Region Code: FE
               Video Region: 1
               Features: 1

                       Feature 4 --------
                       Feature title: Star!
                       Length: 2:52:51
                       Aspect ratio: 2.20:1 [STANDARD]
                       Primary genre: Comedy
                       Rating: G [MPAA (USA)]
                       Feature type: Main
                       Production type: Motion Picture
                       chapters: 44
                               1: Overture [0:04:19]
                               2: Main Titles [0:01:28]
                               3: A Poor Start [0:04:13]
                               4: Piccadilly [0:04:06]
                               5: Bon Voyage [0:01:25]
                               6: Oh, It's A Lovely War [0:02:02]
                               7: Bloody Awful [0:01:38]
                               8: In My Garden Of Joy [0:02:04]
                               9: London! [0:02:08]
                               10: Forbidden Fruit [0:01:47]
                               11: An Original Entrance [0:03:28]
                               12: 'n' Everything [0:02:17]
                               13: Love And Marriage [0:04:12]
                               14: Burlington Bertie From Bow [0:04:21]
                               15: The Knight Returns [0:08:15]
                               16: An Uncrowned Queen [0:07:14]
                               17: Parisian Pierrot [0:03:17]
                               18: Naked Men, Naked Women [0:03:05]
                               19: America! [0:05:20]
                               20: Limehouse Blues [0:08:19]
                               21: Ruling The Yanks [0:06:59]
                               22: Someone To Watch Over Me [0:03:16]
                               23: Toga Party [0:02:43]
                               24: Dear Little Boy (Dear Little Girl) [0:01:57]
                               25: Who Am I? [0:03:12]
                               26: On Holiday [0:06:26]
                               27: Not-So-Private Lives [0:04:09]
```

```
                              28: Someday I'll Find You [0:01:20]
                              29: Safety In Numbers [0:04:42]
                              30: Farewell, Kind Sir [0:05:07]
                              31: Witness For The Defense [0:04:14]
                              32: Physician [0:04:22]
                              33: I'll Show You! [0:01:06]
                              34: Do Do Do [0:02:59]
                              35: What Do You Want? [0:03:07]
                              36: Has Anybody Seen Our Ship? [0:02:55]
                              37: The Act Continues [0:06:33]
                              38: Faultless Timing [0:10:27]
                              39: Skylark [0:04:52]
                              40: My Ship [0:02:36]
                              41: Stop Acting! [0:03:06]
                              42: The Saga Of Jenny [0:07:05]
                              43: Love And Marriage II [0:01:21]
                              44: End Credits [0:03:01]
      Side 2 --------
      Number layers: 1
              Layer 1 --------
              Media type: DVD
              TV system: NTSC
              Region Code: FE
              Video Region: 1
              Features: 3

                      Feature 2 --------
                      Feature title: Star! The Sound Of A Legend
                      Length: 0:10:53
                      Aspect ratio: 2.35:1 [ANAMORPHIC]
                      Primary genre: Comedy
                      Rating: G [MPAA (USA)]
                      Feature type: Main
                      Production type: Motion Picture
                      chapters: 0

                      Feature 9 --------
                      Feature title: Silver Star
                      Length: 0:09:04
                      Primary genre: Documentary
                      Feature type: Extra
                      chapters: 0

                      Feature 10 --------
                      Feature title: Julie Andrews And Daniel Massey Screen Test
                      Length: 0:10:36
                      Aspect ratio: 2.35:1 [ANAMORPHIC]
                      Primary genre: Comedy
                      Rating: G [MPAA (USA)]
                      Feature type: Main
                      Production type: Motion Picture
                      chapters: 0
```

Video Metadata Reference

Video Metadata Reference Overview

The following metadata references are intended to give the reader some idea of what is returned from Video Query APIs. They were generated using the sample applications in the "Commonly Used Video Query APIs" section.

Contributor Metadata Example

The following is intended to give the reader some idea of the metadata returned in a gnsdk_video_query_find_contributors() API call. The output was generated using the sample code in the "Find Contributors" article.

```
<CONTRIBUTOR_RESPONSE>
  <RANGE_START>1</RANGE_START>
  <RANGE_END>10</RANGE_END>
  <RANGE_TOTAL>43</RANGE_TOTAL>
  <CONTRIBUTOR ORD="1">
    <NAME_OFFICIAL>
      <DISPLAY>Christian Bale</DISPLAY>
    </NAME_OFFICIAL>
    <BIOGRAPHY LANG="eng">Welsh actor Christian Bale first drew attention on the silver screen when
he was
    13 for his role as a English boy trapped in a Japanese internment camp in the World War II drama
Empire
    of the Sun. After starring in the Disney musical film Newsies, he was cast opposite Winona Ryder
in
    Little Women; ...(more)
    </BIOGRAPHY>
    <BIRTH_DATE ORD="1">1974-01-30</BIRTH_DATE>
    <BIRTH_PLACE ORD="1">Haverfordwest, Wales, United Kingdom</BIRTH_PLACE>
    <CREDIT>
      <NAME_OFFICIAL>
        <DISPLAY>Christian Bale</DISPLAY>
      </NAME_OFFICIAL>
      <CHARACTER>Batman</CHARACTER>
      <RANK>1</RANK>
      <CONTRIBUTOR>
        <NAME_OFFICIAL>
          <DISPLAY>Christian Bale</DISPLAY>
        </NAME_OFFICIAL>
      </CONTRIBUTOR>
      <VIDEO_WORK ORD="1">
        <DATE_ORIGINAL>2012-07-20</DATE_ORIGINAL>
      </VIDEO_WORK>
    </CREDIT>

    ....More Credits

  </CONTRIBUTOR>
  <CONTRIBUTOR ORD="2">
    <NAME_OFFICIAL>
      <DISPLAY>Heath Ledger</DISPLAY>
    </NAME_OFFICIAL>
    <BIOGRAPHY LANG="eng">The death of actor Heath Ledger in 2008, at age 28, sent shock waves
through the
    film world, particularly among his legions of fans who had only just discovered his intense
screen charisma.
    Having grown up in Perth, Australia, Ledger waited until he finished high school before setting
out on a path
    that would lead him quickly to Hollywood success. ...(more)
    </BIOGRAPHY>
    <BIRTH_DATE ORD="1">1979-04-04</BIRTH_DATE>
    <BIRTH_PLACE ORD="1">Perth, Western Australia, Australia</BIRTH_PLACE>
    <DEATH_DATE ORD="1">2008-01-22</DEATH_DATE>
    <DEATH_PLACE ORD="1">New York City, New York, United States</DEATH_PLACE>
    <CREDIT>
      <NAME_OFFICIAL>
        <DISPLAY>Heath Ledger</DISPLAY>
      </NAME_OFFICIAL>
      <CHARACTER>Tony</CHARACTER>
      <RANK>1</RANK>
      <CONTRIBUTOR>
        <NAME_OFFICIAL>
          <DISPLAY>Heath Ledger</DISPLAY>
        </NAME_OFFICIAL>
      </CONTRIBUTOR>
      <VIDEO_WORK ORD="1">
        <DATE_ORIGINAL>2009</DATE_ORIGINAL>
      </VIDEO_WORK>
    </CREDIT>

    ... More Credits

    </CONTRIBUTOR>

    ... More Contributors

 </CONTRIBUTOR_RESPONSE>
```

Product Metadata Example

The following is intended to give the reader some idea of the metadata returned in a gnsdk_video_query_find_products() API call. The output was generated using the sample code in the "Find Products" article.

```
<VIDEO_PRODUCT>
  <PACKAGE_LANG ID="1" LANG="eng">English</PACKAGE_LANG>
  <PRODUCTION_TYPE ID="16635">Motion Picture</PRODUCTION_TYPE>
  <TITLE_OFFICIAL>
    <DISPLAY>A Bug&apos;s Life</DISPLAY>
    <MAIN_TITLE>A Bug&apos;s Life</MAIN_TITLE>
  </TITLE_OFFICIAL>
  <RATING TYPE="MPAA (USA)" TYPE_ID="1" ID="1">G</RATING>
  <DATE_RELEASE>2000-08-01</DATE_RELEASE>
  <DATE_ORIGINAL>1995-11-20</DATE_ORIGINAL>
  <ASPECT_RATIO TYPE="STANDARD">2.35:1</ASPECT_RATIO>
  <VIDEO_REGION ID="2">1</VIDEO_REGION>
  <DURATION UNITS="sec">5700</DURATION>
  <VIDEO_DISC ORD="1">
```

```
    <DISC_NUM>1</DISC_NUM>
    <VIDEO_SIDE ORD="1">
      <SIDE_NUM>1</SIDE_NUM>
      <VIDEO_LAYER ORD="1">
        <LAYER_NUM>1</LAYER_NUM>
        <TV_SYSTEM>NTSC</TV_SYSTEM>
        <REGION_CODE>FE</REGION_CODE>
        <VIDEO_REGION ID="2">1</VIDEO_REGION>
        <MEDIA_TYPE ID="2">DVD</MEDIA_TYPE>
        <VIDEO_FEATURE ORD="1">
          <FEATURE_NUM>4</FEATURE_NUM>
          <FEATURE_TYPE ID="1">Main</FEATURE_TYPE>
          <PRODUCTION_TYPE ID="16635">Motion Picture</PRODUCTION_TYPE>
          <TITLE_OFFICIAL>
            <DISPLAY>A Bug&apos;s Life [Full Screen Version]</DISPLAY>
          </TITLE_OFFICIAL>
          <ASPECT_RATIO TYPE="STANDARD">2.35:1</ASPECT_RATIO>
          <DURATION UNITS="sec">5688</DURATION>
          <GENRE_LEVEL1>Action/Adventure</GENRE_LEVEL1>
          <GENRE_LEVEL2>Adventure</GENRE_LEVEL2>
          <RATING TYPE="MPAA (USA)" TYPE_ID="1" ID="1">G</RATING>
          <CHAPTER ORD="1">
            <CHAPTER_NUM>1</CHAPTER_NUM>
            <TITLE_OFFICIAL>
              <DISPLAY>Program Start</DISPLAY>
            </TITLE_OFFICIAL>
            <DURATION UNITS="sec">27</DURATION>
          </CHAPTER>
          <CHAPTER ORD="2">
            <CHAPTER_NUM>2</CHAPTER_NUM>
            <TITLE_OFFICIAL>
              <DISPLAY>Main Titles</DISPLAY>
            </TITLE_OFFICIAL>
            <DURATION UNITS="sec">241</DURATION>
          </CHAPTER>

          ...More Chapters


        </VIDEO_FEATURE>

        ... More Video Features

      </VIDEO_LAYER>
    </VIDEO_SIDE>
  </VIDEO_DISC>
</VIDEO_PRODUCT>
```

Work Metadata Example

The following is intended to give the reader some idea of the metadata returned in a gnsdk_video_query_find_works() API call. The output was generated using the sample code in the "Find Works" article.

```
<VIDEO_WORK>
  <TITLE_OFFICIAL LANG="eng">
```

```
    <DISPLAY>Cowboys &amp; Aliens</DISPLAY>
  </TITLE_OFFICIAL>
  <ORIGIN_LEVEL1 ID="4123">North America</ORIGIN_LEVEL1>
  <ORIGIN_LEVEL2 ID="4123">United States</ORIGIN_LEVEL2>
  <ORIGIN_LEVEL3 ID="4123">United States</ORIGIN_LEVEL3>
  <ORIGIN_LEVEL4 ID="4123">United States</ORIGIN_LEVEL4>
  <GENRE_LEVEL1>Action/Adventure</GENRE_LEVEL1>
  <GENRE_LEVEL2>Action</GENRE_LEVEL2>
  <PRODUCTION_TYPE ID="16635">Motion Picture</PRODUCTION_TYPE>
  <DATE_ORIGINAL>2011-07-29</DATE_ORIGINAL>
  <DURATION UNITS="SEC">8119</DURATION>
  <PLOT_SYNOPSIS LANG="eng">Bearing a mysterious metal shackle on his wrist, an amnesiac
gunslinger (Daniel Craig) wanders into a frontier town called Absolution. He quickly finds that
strangers are unwelcome, and no one does anything without the approval of tyrannical Col. Dolarhyde
(Harrison Ford). But when Absolution faces a threat from beyond our world, the stranger finds he is
their only hope of salvation. He unites townspeople, Dolarhyde and Apache warriors against the alien
forces in a epic battle for survival.</PLOT_SYNOPSIS>
  <RATING TYPE="MPAA (USA)" TYPE_ID="1" REASON="Some Partial Nudity, A Brief Crude Reference,
Western and Sci-Fi Action, Western and Sci-Fi Violence" ID="3">PG-13</RATING>
  <RATING TYPE="BBFC (UK)" TYPE_ID="6" REASON="Scenes of Intense Threat, Moderate Action Violence"
ID="36">12A</RATING>
  <RATING TYPE="OFLC (Australia)" TYPE_ID="8" ID="46">M15+</RATING>
  <RATING TYPE="RÃ©gie du cinÃ©ma (QuÃ©bec)" TYPE_ID="16" ID="90">G</RATING>
  <RATING TYPE="DJTCQ (Brazil)" TYPE_ID="15" ID="85">12</RATING>
  <RATING TYPE="RÃ©gie du cinÃ©ma (QuÃ©bec)" TYPE_ID="16" REASON="Violence" ID="91">13+</RATING>
  <RATING TYPE="MCC (France)" TYPE_ID="17" ID="94">U</RATING>
  <RATING TYPE="FSK (Germany)" TYPE_ID="18" ID="102">12</RATING>
  <RATING TYPE="CRC (Italy)" TYPE_ID="19" ID="105">T</RATING>
  <WORK_TYPE ID="2">Audio-Visual</WORK_TYPE>
  <COLOR_TYPE ID="2">Color</COLOR_TYPE>
  <AUDIENCE ID="61205">Middle-Aged</AUDIENCE>
  <MOOD ID="60973">Dazzling</MOOD>
  <STORY_TYPE ID="61969">Against The Odds</STORY_TYPE>
  <SCENARIO ID="61109">Alien Invasion</SCENARIO>
  <SETTING_ENVIRONMENT ID="61890">Bar / Pub</SETTING_ENVIRONMENT>
  <SETTING_TIME_PERIOD ID="23846">1870&apos;s</SETTING_TIME_PERIOD>
  <SOURCE ID="61849">Comic Book</SOURCE>
  <STYLE ID="61173">Blockbuster</STYLE>
  <TOPIC ID="61702">Aliens</TOPIC>
  <CREDIT>
    <ROLE ID="15942">Actor</ROLE>
    <CHARACTER>Jake Lonergan</CHARACTER>
    <RANK>1</RANK>
    <CONTRIBUTOR>
      <NAME_OFFICIAL>
        <DISPLAY>Daniel Craig</DISPLAY>
      </NAME_OFFICIAL>
    </CONTRIBUTOR>
  </CREDIT>
  <CREDIT>
    <ROLE ID="15942">Actor</ROLE>
    <CHARACTER>Woodrow Dolarhyde</CHARACTER>
    <RANK>2</RANK>
    <CONTRIBUTOR>
      <NAME_OFFICIAL>
        <DISPLAY>Harrison Ford</DISPLAY>
      </NAME_OFFICIAL>
    </CONTRIBUTOR>
  </CREDIT>
  <CREDIT>
```

```
    <ROLE ID="15942">Actor</ROLE>
    <CHARACTER>Ella Swenson</CHARACTER>
    <RANK>3</RANK>
    <CONTRIBUTOR>
      <NAME_OFFICIAL>
        <DISPLAY>Olivia Wilde</DISPLAY>
      </NAME_OFFICIAL>
    </CONTRIBUTOR>
  </CREDIT>

  ... More Credits

  <VIDEO_PRODUCT ORD="1">
    <PACKAGE_LANG ID="1" LANG="eng">English</PACKAGE_LANG>
    <TITLE_OFFICIAL>
      <DISPLAY>Cowboys &amp; Aliens</DISPLAY>
      <MAIN_TITLE>Cowboys &amp; Aliens</MAIN_TITLE>
      <EDITION>Extended Edition</EDITION>
    </TITLE_OFFICIAL>
    <ASPECT_RATIO TYPE="ANAMORPHIC">2.40:1</ASPECT_RATIO>
  </VIDEO_PRODUCT>
  <VIDEO_PRODUCT ORD="2">
    <PACKAGE_LANG ID="1" LANG="eng">English</PACKAGE_LANG>
    <TITLE_OFFICIAL>
      <DISPLAY>Cowboys &amp; Aliens</DISPLAY>
      <MAIN_TITLE>Cowboys &amp; Aliens</MAIN_TITLE>
    </TITLE_OFFICIAL>
    <ASPECT_RATIO TYPE="ANAMORPHIC">2.40:1</ASPECT_RATIO>
  </VIDEO_PRODUCT>
  <VIDEO_PRODUCT ORD="3">
    <PACKAGE_LANG ID="30" LANG="ger">German</PACKAGE_LANG>
    <TITLE_OFFICIAL>
      <DISPLAY>Cowboys &amp; Aliens</DISPLAY>
      <MAIN_TITLE>Cowboys &amp; Aliens</MAIN_TITLE>
      <EDITION>Blu-ray + DVD + Digital Copy - Extended Director&apos;s Cut</EDITION>
    </TITLE_OFFICIAL>
    <ASPECT_RATIO TYPE="ANAMORPHIC">2.40:1</ASPECT_RATIO>
  </VIDEO_PRODUCT>

  ... More Video Products

 </VIDEO_WORK>
```

Seasons Metadata Example

The following is intended to give the reader some idea of the metadata returned in a gnsdk_video_query_find_seasons() API call. The output was generated using the sample code in the "Find Seasons" article.

```
<VIDEO_SEASON>
  <TITLE_OFFICIAL LANG="eng">
    <DISPLAY>Season 1</DISPLAY>
  </TITLE_OFFICIAL>
  <DATE_ORIGINAL>1990-12-22</DATE_ORIGINAL>
  <SEASON_NUMBER>1</SEASON_NUMBER>
  <SEASON_COUNT>30</SEASON_COUNT>
  <VIDEO_WORK ORD="1">
```

```
    <TITLE_OFFICIAL LANG="eng" ORD="1">
      <DISPLAY>Simpsons Roasting on an Open Fire</DISPLAY>
    </TITLE_OFFICIAL>
    <SERIES_EPISODE>1</SERIES_EPISODE>
    <SERIES_EPISODE_COUNT>525</SERIES_EPISODE_COUNT>
    <SEASON_EPISODE>1</SEASON_EPISODE>
    <SEASON_EPISODE_COUNT>13</SEASON_EPISODE_COUNT>
  </VIDEO_WORK>
  <VIDEO_WORK ORD="2">
    <TITLE_OFFICIAL LANG="eng" ORD="1">
      <DISPLAY>Bart the Genius</DISPLAY>
    </TITLE_OFFICIAL>
    <SERIES_EPISODE>2</SERIES_EPISODE>
    <SERIES_EPISODE_COUNT>525</SERIES_EPISODE_COUNT>
    <SEASON_EPISODE>2</SEASON_EPISODE>
    <SEASON_EPISODE_COUNT>13</SEASON_EPISODE_COUNT>
  </VIDEO_WORK>
  <VIDEO_WORK ORD="3">
    <TITLE_OFFICIAL LANG="eng" ORD="1">
      <DISPLAY>Homer&apos;s Odyssey</DISPLAY>
    </TITLE_OFFICIAL>
    <SERIES_EPISODE>3</SERIES_EPISODE>
    <SERIES_EPISODE_COUNT>525</SERIES_EPISODE_COUNT>
    <SEASON_EPISODE>3</SEASON_EPISODE>
    <SEASON_EPISODE_COUNT>13</SEASON_EPISODE_COUNT>
  </VIDEO_WORK>

  ... More Video Works

  <VIDEO_SERIES ORD="1">
    <TITLE_OFFICIAL LANG="eng">
      <DISPLAY>The Simpsons</DISPLAY>
    </TITLE_OFFICIAL>
  </VIDEO_SERIES>
</VIDEO_SEASON>
```

Series Metadata Example

The following is intended to give the reader some idea of the metadata returned in a gnsdk_video_query_ find_series() API call. The output was generated using the sample code in the "Find Series" article.

```
<VIDEO_SERIES>
  <TITLE_OFFICIAL LANG="eng">
    <DISPLAY>The Simpsons</DISPLAY>
  </TITLE_OFFICIAL>
  <GENRE_LEVEL1>Animation</GENRE_LEVEL1>
  <GENRE_LEVEL2>Animation</GENRE_LEVEL2>
  <PRODUCTION_TYPE ID="16641">TV Series</PRODUCTION_TYPE>
  <DATE_ORIGINAL>1989-12-17</DATE_ORIGINAL>
  <PLOT_SYNOPSIS LANG="eng">Homer and Marge Simpson raise Bart, Lisa and baby Maggie.</PLOT_
SYNOPSIS>
  <SERIAL_TYPE ID="3">Series TV</SERIAL_TYPE>
  <WORK_TYPE ID="2">Audio-Visual</WORK_TYPE>
  <COLOR_TYPE ID="2">Color</COLOR_TYPE>
  <AUDIENCE ID="61201">Tween</AUDIENCE>
  <MOOD ID="61041">Goofy</MOOD>
```

```
<STORY_TYPE ID="60743">Family Saga</STORY_TYPE>
<REPUTATION ID="61189">Critical Darling</REPUTATION>
<SCENARIO ID="61069">Bumbling Buffoons</SCENARIO>
<SETTING_ENVIRONMENT ID="60786">Suburbia</SETTING_ENVIRONMENT>
<SETTING_TIME_PERIOD ID="2645">1990&apos;s</SETTING_TIME_PERIOD>
<SOURCE ID="61856">Original Screenplay</SOURCE>
<STYLE ID="62014">Traditional Animation</STYLE>
<TOPIC ID="61468">Family Dysfunction</TOPIC>
<VIDEO_SEASON ORD="1">
  <TITLE_OFFICIAL LANG="eng">
    <DISPLAY>Season 1</DISPLAY>
  </TITLE_OFFICIAL>
  <SEASON_NUMBER>1</SEASON_NUMBER>
  <SEASON_COUNT>29</SEASON_COUNT>
</VIDEO_SEASON>
<VIDEO_SEASON ORD="2">
  <TITLE_OFFICIAL LANG="eng">
    <DISPLAY>Season 2</DISPLAY>
  </TITLE_OFFICIAL>
  <SEASON_NUMBER>2</SEASON_NUMBER>
  <SEASON_COUNT>29</SEASON_COUNT>
</VIDEO_SEASON>
<VIDEO_SEASON ORD="3">
  <TITLE_OFFICIAL LANG="eng">
    <DISPLAY>Season 3</DISPLAY>
  </TITLE_OFFICIAL>
  <SEASON_NUMBER>3</SEASON_NUMBER>
  <SEASON_COUNT>29</SEASON_COUNT>
</VIDEO_SEASON>

... More Video Seasons

<CREDIT>
  <CHARACTER>Homer Simpson</CHARACTER>
  <RANK>1</RANK>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Dan Castellaneta</DISPLAY>
    </NAME_OFFICIAL>
  </CONTRIBUTOR>
</CREDIT>
<CREDIT>
  <CHARACTER>Marge Simpson</CHARACTER>
  <RANK>2</RANK>
  <CONTRIBUTOR>
    <NAME_OFFICIAL>
      <DISPLAY>Julie Kavner</DISPLAY>
    </NAME_OFFICIAL>
  </CONTRIBUTOR>
</CREDIT>

... More Credits

</VIDEO_SERIES>
```

Suggestions Metadata Example

The following is intended to give the reader some idea of the metadata returned in a gnsdk_video_query_ find_suggestions() API call. The output was generated using the sample code in the "Find Suggestions" article.

```
<SUGGESTIONS_RESPONSE>
  <RANGE_START>1</RANGE_START>
  <RANGE_END>20</RANGE_END>
  <RANGE_TOTAL>56</RANGE_TOTAL>
  <TEXT TYPE="DVDCASE" ORD="1">Spider-Man 1</TEXT>
  <TEXT TYPE="DVDCASE" ORD="2">Spider-Man</TEXT>
  <TEXT TYPE="DVDCASE" ORD="3">Spider-Man Trilogy: Spider-Man / Spider-Man 2 / Spider-Man 3</TEXT>
  <TEXT TYPE="DVDCASE" ORD="4">Spider-Man 3</TEXT>
  <TEXT TYPE="DVDCASE" ORD="5">Spider-Man 2</TEXT>
  <TEXT TYPE="DVDCASE" ORD="6">The Spiderwick Chronicles</TEXT>
  <TEXT TYPE="DVDCASE" ORD="7">spider-Man - The Venom Saga</TEXT>
  <TEXT TYPE="DVDCASE" ORD="8">Spider Baby</TEXT>
  <TEXT TYPE="DVDCASE" ORD="9">Spider-Man: Die High Definition Trilogie</TEXT>
  <TEXT TYPE="DVDCASE" ORD="10">Spider-Man: The &apos;67 Classic Collection</TEXT>
  <TEXT TYPE="DVDCASE" ORD="11">Spider-Man: The High Definition Trilogy</TEXT>
  <TEXT TYPE="DVDCASE" ORD="12">Spider-Man Vs. Doc Ock</TEXT>
  <TEXT TYPE="DVDCASE" ORD="13">Spider&apos;s Web</TEXT>
  <TEXT TYPE="DVDCASE" ORD="14">Spider</TEXT>
  <TEXT TYPE="DVDCASE" ORD="15">Spider-Man: The Ultimate Villain Showdown</TEXT>
  <TEXT TYPE="DVDCASE" ORD="16">Spider-Man: La TrilogÃa</TEXT>
  <TEXT TYPE="DVDCASE" ORD="17">Spider-Man: The Return Of The Green Goblin</TEXT>
  <TEXT TYPE="DVDCASE" ORD="18">Spiders</TEXT>
  <TEXT TYPE="DVDCASE" ORD="19">The Spiders</TEXT>
  <TEXT TYPE="DVDCASE" ORD="20">Spider-Man: The New Animated Series</TEXT>
</SUGGESTIONS_RESPONSE>
```

Accessing Enriched Content (Link)

Using Link

Besides acquiring the necessary licensing, including 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();
```

Accessing Enriched Music Content

To access enriched metadata content, you can purchase additional metadata entitlements and use the Link module APIs in your application. For more information, see "Link Module Overview" on page 26.

You can use the Link module to retrieve enriched content, including:

- Cover art
- Artist images
- Genre art

If there was no match to Gracenote content for a query, you can use the default cover art symbol instead. For more information, see "Using a Default Image When Cover Art Is Missing" on page 215.

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 for Desktop 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 "Music Recognition Overview" on page 108.

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 "Lookup Modes" on page 79 and "Using SQLite for Storage and Caching" on page 96.

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 for Desktop recognition feature, such as MusicID or MusicID-File:

```
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" on page 27.

```
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" on page 165, 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" on page 165, 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 for Desktop 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 for Desktop tries to retrieve. If an image of that size isn't found, the second size will be the next size that GNSDK for Desktop tries to retrieve, and so on. For example, the following code calls the gnsdk_link_query_option_set() function three times, so that GNSDK for Desktop 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

Related Information

Enriched Content Module (Link) 26

Accessing Enriched Video Content

The general process to access enriched video metadata is:

1. Use gnsdk_video_query_option_set() with the GNSDK_VIDEO_OPTION_ENABLE_LINK_ DATA option to set the query to enable access to Link metadata.
2. Pass in a source GDO.
3. Pass in any necessary options, such as a track ordinal number or an image size.
4. Access the desired content using gnsdk_link_query_content_retrieve().
5. When an application has finished using the content, free the content using gnsdk_link_query_ content_free().

Sample Code:

video_link_lookup/main.c

Application steps:

1. Authenticate caller, initialize GNSDK and Video module, enable SDK logging, load a locale, and get a user handle
2. Perform find Products search using TOC
3. Select first match and parse/display metadata.
4. Create a link query
5. Run query and access and display cover art
6. Release resources and shutdown GNSDK and Video module

Sample output:

```
GNSDK Product Version   : 3.2.0.274      (built 2012-11-30 09:09-0800)

Title: A Bug's Life

Thumbnail cover art: 99672 byte JPEG
```

Improving Retrieval Performance by Using Enriched Content URLs

There may be cases when you wish to retrieve metadata, descriptors, or external identifiers as quickly as possible, while lazy-loading enriched content such as cover art in the background. To achieve this, 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.

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.

Option	Value
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.
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_*.
GNSDK_GDO_VALUE_URL_TVCHANNEL_ IMAGE_*	Retrieves the temporary URL for the contributor image. Sizes available are 110, 110 or larger, 220, and 220 or smaller.

Implementing Discovery Features

Implementing Playlist Features

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 180.

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 for Desktop 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 results returned.

Option	Description
GNSDK_ PLAYLIST_ MORELIKETHIS_ OPTION_MAX_ PER_ARTIST	The maximum number of results per artist returned.
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.
GNSDK_ PLAYLIST_ MORELIKETHIS_ OPTION_ RANDOM	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). 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.

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_RANDOM,
    "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 for Desktop 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.

Synchronization of a collection summary to physical media involves two steps:

1. Adding all existing (current and new) unique identifiers, using the gnsdk_playlist_collection_sync_ ident_add() function, which Playlist collates.
2. Calling 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.

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_RANDOM :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_RANDOM :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
```

Related Information

Discovery Features 19

Playlist PDL Specification

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

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

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

PDL Syntax

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

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

Keywords

PDL supports these keywords:

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

Keyword	Description	Required or Optional	PDL Statement Order
ORDER	Specifies the criteria used to order the results' display. If a PDL statement does not include the ORDER keyword, Playlist returns results in random order. Example: Display results in based on a calculated similarity value; tracks having greater similarity values to input criteria display higher in the results. The expression format is: <identifier> <operator> <identifier>	Optional	3
LIMIT	Specifies criteria used to restrict the number of returned results. Also uses the keywords RESULT and PER. Example: Limiting the number of tracks displayed in a playlist to 30 results with a maximum of two tracks per artist. The expression format is: <identifier> <operator> <identifier>	Optional	4
SEED	Specifies input data criteria from one or more idents. Typically, a Seed is the This in a More Like This playlist request. Example: Using a Seed of Norah Jones' track Don't Know Why to generate a playlist of female artists of a similar genre. The expression format is: <identifier> <operator> <identifier>	Optional	NA

Example: Keyword Syntax

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

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

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

Operators

PDL supports these operators:

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

Literals

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

- Single quotes: 'value one'
- Double quotes: "value two"
- Single quotes surrounded by double quotes: "'value three'"

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

Attributes

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

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

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

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

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

Functions

PDL supports these functions:

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

PDL Statements

This topic discusses PDL statements and their components.

Attribute Implementation <att_imp>

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

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

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

GN_ArtistName == "ACDC"

or

"ACDC" == GN_ArtistName

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

"GENERATE PLAYLIST WHERE LIKE SEED > 500"

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

Expression <expr>

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

[<number> <math_op>] <att_imp>

For example:

3 * (GN_Era LIKE SEED)

Score <score>

Scores can range between -1000 and 1000.

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

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

Example: PDL Statements

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

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

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

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

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

- ACDC
- AC/DC
- AC*DC

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

The following PDL example generates results where:

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

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

Implementing MoodGrid

The MoodGrid APIs:

- Encapsulate Gracenote’s Mood Editorial Content (mood layout and ids).
- Simplify access to MoodGrid results through x,y coordinates.
- Allow for multiple local and online data sources through MoodGrid Providers.

- Enable pre-filtering of results using genre, origin, and era attributes.
- Support 5x5 or 10x10 MoodGrids.
- Provide the ability to go from a cell of a 5x5 MoodGrid to any of its expanded four Moods in a 10x10 grid.

Implementing MoodGrid in an application involves the following steps:

1. Initializing the MoodGrid module.
2. Enumerating the data sources using MoodGrid Providers.
3. Creating and populating a MoodGrid Presentation.
4. Filtering the results, if needed.

Initializing the MoodGrid Module

Before using the MoodGrid APIs, follow the usual steps to initialize GNSDK for Desktop. The following GNSDK for Desktop modules must be initialized:

- GNSDK Manager
- SQLite (for local caching)
- MusicID
- Playlist
- MoodGrid

For more information on initializing the GNSDK for Desktop modules, see “Initializing an Application.”

To initialize MoodGrid, use the gnsdk_moodgrid_initialize() function.

```
error = gnsdk_moodgrid_initialize(sdkmgr_handle);
```

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 MoodGrid will return no results. Please contact your Gracenote Global Services & Support representative for more information.

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 MoodGrid will return no results.

Enumerating Data Sources using MoodGrid Providers

GNSDK automatically registers all local and online data sources available to MoodGrid. For example, if you create a playlist collection using the Playlist API, GNSDK for Desktop automatically registers that playlist as a data source available to MoodGrid. These data sources are referred to as Providers. MoodGrid is designed to work with multiple providers. You can iterate through the list of available Providers using the 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:

1. Name (GNSDK_MOODGRID_PROVIDER_NAME key)
2. Type (GNSDK_MOODGRID_PROVIDER_TYPE key)
3. Network Use (GNSDK_MOODGRID_PROVIDER_NETWORK_USE key)

Creating and Populating a MoodGrid Presentation

Once you have a handle to a Provider, you can create and populate a MoodGrid Presentation with Mood data. A Presentation is a data structure that represents the MoodGrid, containing the mood name and playlist information associated with each cell of the grid.

To create a MoodGrid Presentation, use the gnsdk_moodgrid_presentation_create() function, passing in the user handle, the type of the MoodGrid, 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 an 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 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 MoodGrid Results

If you wish, you can filter the MoodGrid results. MoodGrid 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 MoodGrid

When you are finished using MoodGrid, you can use the gnsdk_moodgrid_shutdown() function to shut it down. The GNSDK for Desktop modules necessary for running MoodGrid should be shut down in the following order:

1. MoodGrid
2. Playlist
3. MusicID
4. SQLite
5. GNSDK Manager

Example: Working with MoodGrid

This example provides a complete MoodGrid 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
```

Related Information

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 for Desktop. The following GNSDK for Desktop 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 for Desktop modules, see “Initializing an Application.”

To initialize Rhythm, use the gnsdk_rhythm_initialize() function.

```
error = gnsdk_rhythm_initialize(sdkmgr_handle);
```

If you are using MusicID to recognize music, you must enable Playlist and DSP data in your query.

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 (e.g., Amazon)
- **GNSDK_RHYTHM_OPTION_ENABLE_CONTENT_DATA** - Response should include data for fetching content (e.g., 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 for Desktop 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.

Submitting Content to Gracenote

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.

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.

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.

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: • gnsdk_submit_parcel_create() • gnsdk_submit_parcel_upload() • gnsdk_submit_parcel_release ()	Submit, Parcel Submit Requirements

Editable GDOs

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_ALBUM_COMPILATION 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)

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.)

Editable GDO Considerations

When creating editable GDOs, consider these guidelines:

1. 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.
2. 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.

3. 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.

1. 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.

1. 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.

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.

Submitting Metadata

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)

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.

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
gnsdk_submit_parcel_data_init_features()	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.

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.

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.

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.

Submitting Parcels

The Submit APIs for parcels support the following use cases:

1. Add 1-n editable GDOs or 1-n Features, or both to a parcel.

1. Add a unique identifier for a GDO or a Feature used later to access callback status information.

1. Bundle editable GDOs and Features together in a parcel, or upload either item separately.

1. Submit a parcel numerous times.

The following table shows the Submit functions used to create and process parcels.

Function	Usage
gnsdk_submit_parcel_create ()	Create an empty parcel and add editable GDOs, feature data, or both.
gnsdk_submit_parcel_data_ add_gdo()	Add completed editable GDOs to a parcel for Submit uploading.
gnsdk_submit_parcel_upload ()	Upload a parcel containing editable GDOs, feature data, or both to Gracenote Service.
gnsdk_submit_parcel_data_ get_state()	Set callback functionality to access a parcel's upload progress and network status information.

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.

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.

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:

1. GNSDK_GDO_VALUE_TITLE_SORTABLE must stay synchronized with GNSDK_GDO_VALUE_TITLE_DISPLAY in Album and Track types.

1. 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.

1. 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.

Advanced Topics

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.

Identifier	Description	Best Use
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.
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).

Identifier	Description	Best Use
GlobalID	The GlobalID representst 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). Functionality depends on integration with the Nuance VoCon Music Premium SDK.	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.

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

GNSDK for Desktop Identifiers Comparison Matrix

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

Identifier	Pros	Cons
Serialized GDO	Good for getting back to the original metadata. Good for joining responses and requests between different Gracenote products Does not require knowledge of the object type	Cannot be used for comparison to other serialized GDO values Not useful for integrating GNSDK for Desktop to other metadata sources, as only GNSDK for Desktop supports GDOs Relatively large and requires much storage space

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

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 CDDBID

```
gnsdk_gdo_handle_t query_gdo;
gnsdk_cstr_t cddbid = "4+029C475EFCF8D469C78A644DEBFABF91+5795407";
error = gnsdk_manager_gdo_create_from_id(cddbid, GNSDK_NULL, GNSDK_ID_SOURCE_CDDBID, &query_gdo);
if (!error)
error = gnsdk_musicid_query_set_gdo(query_gdo);
```

Example: Creating a GDO from Raw CDDBID

A Raw CDDBID is a decomposition of the CDDBID 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_CDDBID, &query_gdo);
if (!error)
error = gnsdk_musicid_query_set_gdo(query_gdo);
```

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()

Load Balancing

GNSDK for Desktop'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 for Desktop retrieves a set of IP addresses for each hostname it connects to. These IPs include multiple Gracenote Service locations, rather than just a single IP (and likewise, location) as occurs when load balancing is disabled. Once accessed, GNSDK for Desktop 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 for Desktop 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 for Desktop 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 for Desktop immediately expires them and re-retrieves another set.

Implementation Considerations

Gracenote strongly recommends enabling this option for these application scenarios:

1. The application performs long-running processes.

1. 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.

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.
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.

Option	Meaning
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.

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.

Creating a GDO From XML

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

Using the Video TOC Generator APIs

The Gracenote SDK provides APIs for reading TOC (Table of Contents) information from DVD and Blu Ray discs. From this information, the APIs can generate an XML TOC string that you can use to look up the DVD in the Gracenote Service.

To use the TOC generator utility APIs, include the `gnsdk_tocgen.h` header file and call the TOC generator initialization function:

```
#include "gnsdk_tocgen.h"
gnsdk_tocgen_initialize();
```

After initialization, your application needs to make a call to recognize the TOC source. This API takes a path (e.g., "D:") to the device and generates a handle for use in later calls:

```
gnsdk_tocgen_handle_t handle = GNSDK_NULL;
gnsdk_char_t* device_path = "D:";
gnsdk_tocgen_read_source(device_path, GNSDK_TOCGEN_OPTION_FLAG_DEFAULT, &handle);
```

As you can see, the call takes an options define, but, currently, only the default one is available.

Once you have a handle, your application can call the API that generates an XML TOC string, which you can use to query the Gracenote Service:

```
gnsdk_str_t toc = GNSDK_NULL;
gnsdk_tocgen_generate_toc(handle, GNSDK_TOCGEN_TOC_GNXML, &toc);
```

Example generated XML TOC string:

```
<?xml version="1.0" encoding="UTF-8" ?>
<DVD_THIN_TOC VER="1" ANGLED="FALSE">
  <VIDEO_TS>
    <REGION>FE</REGION>
    <VTS_ATTR CNT="22">
      <VTS N="1">
        <VIDEO_ATTR>
          <COMPRESSION_MODE TYPE="1"/>
          <TV_SYSTEM TYPE="0"/>
          <ASPECT_RATIO TYPE="3"/>
          <DISP_MODE TYPE="2"/>
          <CC_FIELD1 TYPE="1"/>
          <CC_FIELD2 TYPE="0"/>
          <SRC_PIC_RESOLUTION TYPE="0"/>
          <SRC_PIC_LETTERBOXED TYPE="0"/>
          <FILM_CAMERA_MODE TYPE="0"/>
        </VIDEO_ATTR>
        <TITLE_UNIT_ARRAY CNT="1">
          <TITLE_UNIT CNT="28" N="1">6389 6865 5693 8331 2280 5786 8492
6553 11184
 12656 6248 8730 4561 5770 7995 3354 4176 5109 5444 10969 6261 4458 5281
13466 4203 6397
 6173 45</TITLE_UNIT>
        </TITLE_UNIT_ARRAY>
      </VTS>

   ... More VTS Elements

    </VTS_ATTR>
  </VIDEO_TS>
</DVD_THIN_TOC>
```

You can use the `gnsdk_tocgen_source_info()` call to return information about the disc:

```
gnsdk_tocgen_source_info(handle, GNSDK_TOCGEN_INFO_KEY_MEDIA_TYPE, &value);
```

The above call returns the media type (e.g., 'DVD'). Other information available includes the disc region, disc book type (e.g., 'DVD-ROM'), if the disc is certified, and directory name. See the API reference for a complete list.

Other TOC generator APIs are available to return the build date and version:

```
gnsdk_tocgen_get_version();     // Return TOC gen version
gnsdk_tocgen_get_build_date();  // Return TOC gen build date
```

Releasing Resources and Shutting Down

When you are done with a particular resource (handle, TOC string) you should release it:

```
gnsdk_tocgen_release(handle);  // Release TOC handle
gnsdk_tocgen_free_toc(toc);    // Release TOC string
```

Last, but not least, you should call the TOC generator shutdown API:

```
gnsdk_tocgen_shutdown();
```

Note: If the initialization call returns an error, **do not** call shutdown.

TOC Generator Sample Utility Program

The GNSDK for Desktop comes with a utility sample program - `/utilities/samples_c/tocgen/main.c` - that illustrates how to use the TOC generator APIs. It generates an XML TOC string.

```
usage:  sample device-path

    device-path   Path to device drive/data
    sample 2TOWERS_D1

Note: If your volume name has spaces then it needs to be in quotes:
      sample "/Volume/Taken 2"
```

Sample Output for DVD 'The Proposal':

```
...\utilities\samples\tocgen> sample D:


TOCGEN Product Version    : 3.05.0.780         (built 2013-04-26 12:29-0700)

Media Type: DVD
Disc Region: FE
Certified: Yes
Disc Book Type: (DVD-ROM)

<?xml version="1.0" encoding="UTF-8" ?>
<DVD_THIN_TOC VER="1" ANGLED="FALSE">
  <VIDEO_TS>
    <REGION>FE</REGION>
    <VTS_ATTR CNT="22">
    </VTS_ATTR>

     ...VTS Elements

  </VIDEO_TS>
</DVD_THIN_TOC>
```

Using Lists

GNSDK for Desktop 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 8.

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 MoodGrid

MoreLikeThis, Playlist, and MoodGrid 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.

List and Locale Interdependence

GNSDK for Desktop 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 99.

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 for Desktop 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 for Desktop 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.

Updating Lists

To update lists, follow the procedure described in "Updating Locales and Lists" on page 105. 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()

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.

Best Practices and Design Requirements

Image Best Practices

Gracenote images – in the form of cover art, artist images and more – are integral features in many online music services, as well as home, automotive and mobile entertainment devices. Gracenote maintains a comprehensive database of images in dimensions to accommodate all popular applications, including a growing catalog of high-resolution (HD) images.

Gracenote carefully curates images to ensure application and device developers are provided with consistently formatted, high quality images – helping streamline integration and optimize the end-user experience. This topic describes concepts and guidelines for Gracenote images including changes to and support for existing image specifications.

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

Image Resizing Guidelines

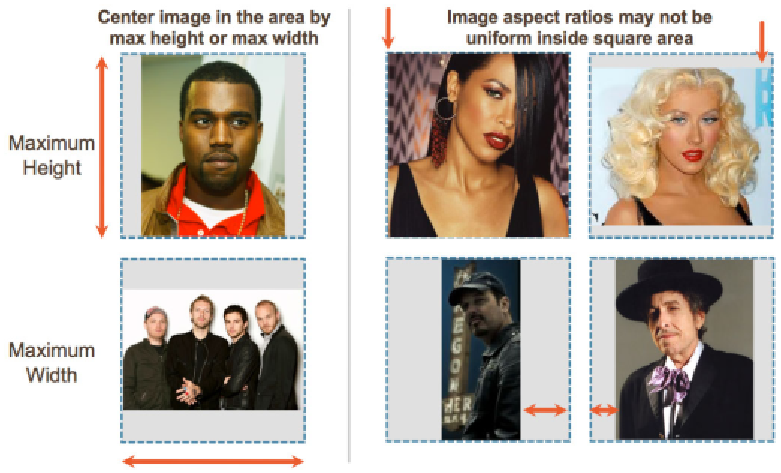
Gracenote images are designed to fi

t within squares as defined by the available image dimensions. This allows developers to present images in a fixed area within application or device user interfaces. Gracenote recommends applications center images horizontally and vertically within the predefined square dimensions, and that the square be transparent such that the background shows through. This results in a consistent presentation despite variation in the image dimensions. To ensure optimum image quality for end-users, Gracenote recommends that applications use Gracenote images in their provided pixel dimensions without stretching or resizing.

Gracenote resizes images based on the following guidelines:

- **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.


Center image in the area by max height or max width
Image aspect ratios may not be uniform inside square area
Maximum Height
Maximum Width

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.

Collaborative Artists Best Practices

The following topic provides best practices for handling collaborations in your application.

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 for Desktop 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.

Displaying Collaborations during Playback

When determining what should be displayed during playback of music, use the following best practices:

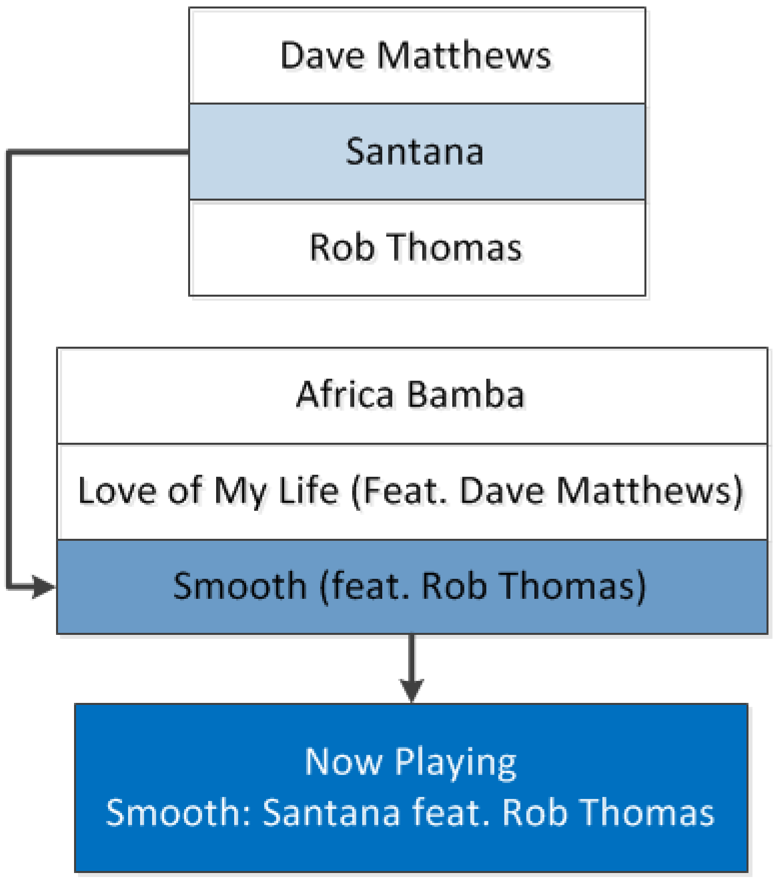
- When a track by a single artist is playing, your application should display the Gracenote normalized text string. For example, when a track by Santana is playing, “Santana” should be displayed.
- When a track by a collaboration is playing, and GNSDK for Desktop has matched the collaboration, the application 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.

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.


Dave Matthews
Santana
Rob Thomas
Africa Bamba
Love of My Life (Feat. Dave Matthews)
Smooth (feat. Rob Thomas)
Now Playing
Smooth: Santana feat. Rob Thomas

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.

Related Information

UI Best Practices for Audio Stream Recognition

The following are recommended best practices for applications that recognize streaming audio. Gracenote periodically conducts analysis on its MusicID-Stream product to evaluate its usage and determine if there are ways we can make it even better. Part of this analysis is determining why some MusicID-Stream recognition queries do not find a match.

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

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

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

Provide Clear and Accessible Instructions

Most failed recognitions are due to incorrect operation by the user. Provide clear and concise instructions to help the user correctly operate the application to result in a higher match rate and a better user experience.For example:

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

Provide a Demo Animation

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

Display a Progress Indicator During Recognition

When listening to audio, the application can receive status updates. The status updates indicate what percentage of the recording is completed. Use this information to display a progress bar (indicator) to notify the user.

Use Animations During Recognition

Display a simple image or animation that shows how to properly perform audio recognition, such as holding the device near the audio source if applicable.

Using Vibration, Tone, or Both to Indicate Recognition Complete

The user may not see visual notifications if they are holding the recording device up to an audio source. Also, the user may pull the device away from an audio source to check if recording has completed. This may result in a poor quality recording.

Display Help Messages for Failed Recognitions

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

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.

C API Reference Documentation

In addition to several popular object-oriented languages, you can develop GNSDK for Desktop applications using C. The following table shows where to find the C API Reference documentation:

API	Description	Location in Package
C API Reference	API descriptions of the GNSDK for DesktopC interface	docs/Content/api_ref_c/html/index.html

C Data Model

The GNSDK for Desktopdata 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 can be found in the online help system provided with GNSDK for Desktop, in the GNSDK for Desktop C API Reference, under GDO Type Mappings.

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 GNSDK API 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 $^{\text{table\_ref}}$ 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.
In Partial	This column lists metadata returned in GDO partial results.
Multi	This column lists elements that can contain multiple results.
Serialized	This column lists metadata returned in serialized GDOs.

GNSDK Glossary

C

Child Key

Used to access other GDOs contained in the current GDO.

E

Editable GDO

A GDO with editable metadata. Only specific fields are editable to preserve the GDO's metadata integrity with Gracenote.

F

Full GDO

A GDO that contains all the information associated with the query match. See Partial GDO.

G

GDO

Gracenote Data Object: Containers most-commonly used to store information returned by a query. GDOs can contain metadata values (names, titles, external IDs, and so on) that can be accessed by an application using Value keys. GDOs can also contain references to other GDOs, which can be accessed using Child GDO keys.

P

Parcel

A Submit container that wraps around editable GDO and feature metadata to enable the metadata 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 modules—the difference is the interaction and end result with Gracenote Service. In most other modules, you create and define queries with specific options and inputs to access metadata from Gracenote Service. In Submit, you create a parcel, add metadata, and upload the parcel to Gracenote Service.

Partial GDO

A GDO that contains a subset of information for the query match, but enough to perform additional processing.

V

Value Key

Used to access metadata Values returned in a GDO.