nielsen

PCM-TO-ID3 SDK

Developer Guide

Release 2.4 Revision N

Revision History

Revision	Date	Description	Author(s)/Editor(s)
A	212-03-06	Initial draft	Fabio Milan Lore Eargle (editor)
B	2012-12-03	Added FAQs, VS 2005 and VS 2008 support, GCC 4.1.2 compilation; addressed 24-bit audio file processing and added new APIs referring to the INFO tag and ID3 tag pass-through processing; incorporated Id3TagParser support	Fabio Milan Lore Eargle (editor) Scott Cooper
C	2013-05-10	Updated for release 1.2	Fabio Milan Lois Price Lore Eargle (editor)
D	2013-11-05	Updated for release version 2.0.0 linked to the latest Id3TagUtils version 3.0.2. Added TI 6424 DSP Appendix	Fabio Milan Lois Price Sudheer Thota Amy Gaither (editor)
E	2014-01-07	Updated for release 2.1	Lois Price Amy Gaither (editor)
F	2014-04-21	Modified pre-parser section to indicate that vendor app may parse as little as 2 minutes of content	Lois Price
G	2014-05-19	Added PCCID and FDCID fields to metadata file, added support for VS 2012	Lois Price
H	2014-06-29	Included in new package. Package changes: added support for VS 2013, made certain that all Microsoft® Windows® project settings are consistent, updated build to 2.1.9	Lois Price

Revision	Date	Description	Author(s)/Editor(s)
I	2014-09-05	Updated the Breakout Code Table, removed all references to ID3 Tag Parser, edited and updated entire document	Shefali Bhat Amy Gaither (editor) Lois Price
J	2015-01-26	Removed references to ID3 Tag Parser, replaced with references to ID3 Tag Validator, improved the explanation of Streaming Modes (0, 1, and 2), added a reference to ID3 Tag Validator.	Lois Price Amy Gaither (editor)
K	2016-04-21	Removed references to Streaming Modes 1 and 2 (no longer supported) and to all related functions, removed conflicting descriptions of supported audio frequencies, reorganized document.	Lois Price Lore Eargle (editor) Amy Gaither (editor)
L	2017-02-08	Removed references to Breakout Codes other than 0 and added mention of MPEG DASH (Dynamic Adaptive Streaming over HTTP)	Lois Price Lore Eargle (editor)
M	2017-03-31	Added MPEG-DASH packaging information	Lois Price Lore Eargle (editor)
N	2017-07-07	Updated for release 2.4	Lois Price Lore Eargle (editor)

Contents

List of Figures

List of Tables

1. Introduction

Important For systems that have used the PCM-to-ID3 SDK releases 2.3 and older, see section 5.7, "SdkTypes.h."

1.1. Purpose

This document describes the concepts and usage of the SDK. It includes requirements and design methods for building applications using the SDK to detect and extract one or more types of Nielsen watermark from uncompressed audio content (pulse code modulation [PCM] audio content), to encrypt watermarks, and to wrap watermarks in ID3 tags for transmission to mobile devices and computer browsers. ID3 tags are Apple's solution for delivering metadata in streaming content.

Most of the guide pertains to the C++ version of the SDK, intended for use on PC operating systems. "Appendix E—DSP Library Implementation" covers C implementation of the SDK for use with the Texas Instruments™ TMS320C6424 Fixed-Point Digital Signal Processor.

1.2. Out of Scope

This document does not describe the technical details of Nielsen ID3 tags. See the *Nielsen ID3 Tag Validator Application User's Guide* for specifics on these tags. If you do not have a copy of that guide, contact your Nielsen technical representative.

Note that Nielsen provides the Nielsen ID3 Tag Validator application, which is a command-line diagnostic tool that is helpful in viewing decrypted ID3 tags and in debugging your application.

1.3. Audience

This guide is intended for experienced C/C++ software developers to use in creating applications based on the Nielsen PCM-to-ID3 Tag SDK (the SDK).

Throughout this document, the term "you" refers to the C++ software developer who is incorporating the Nielsen PCM-to-ID3 SDK into a transcoder application. One function of the transcoder application is to generate Nielsen ID3 tags. This document uses the terms "transcoder" and "ID3-tag generator" interchangeably.

This document focuses primarily on the insertion of Nielsen ID3 tags into HTTP Live-Streaming (HLS) containers. However, PCM-to-ID3 SDK may also be used in a system that generates other types of output (for example, MPEG DASH, which is described in Appendix D—Insert ID3 Tags into an MPEG-DASH Stream).

1.4. References and Related Documents

ANSI/SCTE 104 2012. *Automation System to Compression System Communications Applications Program Interface (API)*. Society of Cable Telecommunications Engineers.

Apple Inc. *Timed Metadata for HTTP Live Streaming*. https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/HTTP_Live_Streaming_Metadata_Spec/Introduction/Introduction.html.

ISO/IEC 13818-1:2007 Information Technology – Generic coding of moving pictures and associated audio information: Systems.

ISO/IEC 23009-1:2014 (E) – Informative Technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats.

Nielsen ID3 Tag Validator SDK User Guide.

Nilsson, M. *ID3 Tag Version 2.4.0 – Main Structure*. http://www.id3.org/id3v2.4.0-structure.

Nilsson, M. *ID3 Tag Version 2.4.0 – Native Frames*. http://www.id3.org/id3v2.4.0-frames.

2. Concept

2.1. System Overview

As shown in Figure 1, your transcoder may be placed one or more points in the distribution chain (including content originator, broadcast affiliate, MVPD/CDN). Regardless of where your ID3-tag generator is placed, it must be downstream of at least one Nielsen audio encoder (a device that inserts Nielsen codes or watermarks into the audio stream). Nielsen ID3 tags are derived from the decoded Nielsen watermarks.

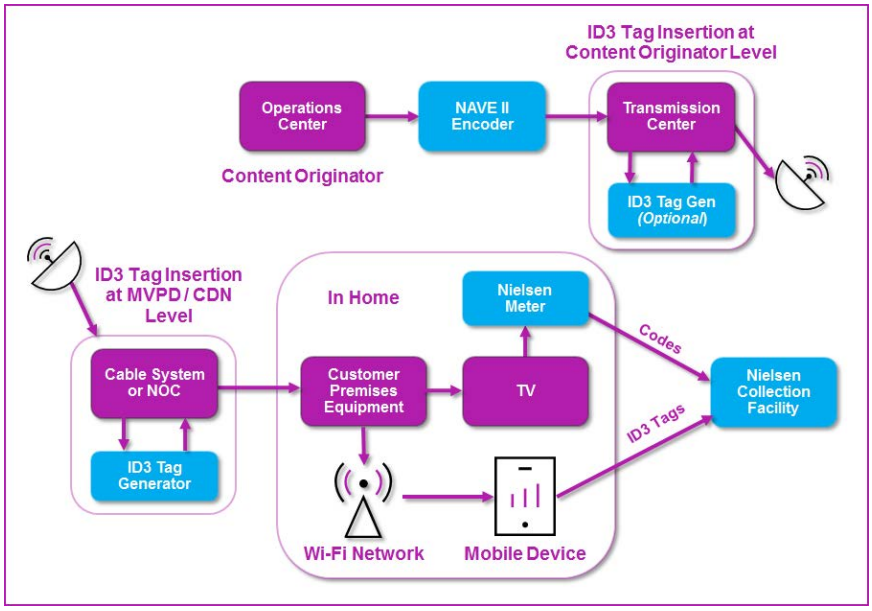

Figure 1 – System Overview

2.2. Transcoder Application Overview

The PCM-to-ID3 SDK libraries that you incorporate into your software include two major components:

- **Nielsen audio decoder**: extracts Nielsen watermarks (NAES 2 and NAES 6) from the PCM audio stream.
- **ID3-tag generator**: creates Nielsen ID3 tags based on the decoded watermarks. Nielsen ID3 tags are similar to Nielsen watermarks, but are in a form that is more easily consumed by mobile devices supporting the Apple HLS (HTTP Live Streaming) specification, MPEG-DASH, and other industry standards.

Your application may include an MPEG-2 multiplexer as well as an HLS segmenter, MPEG-DASH packager, or other packaging software. Alternatively, ID3-tag packaging may be handled by downstream devices. Figure 2 shows the workflow for the generation of HLS streams. Note that the "Packager" might, instead, be an MPEG-DASH packager.

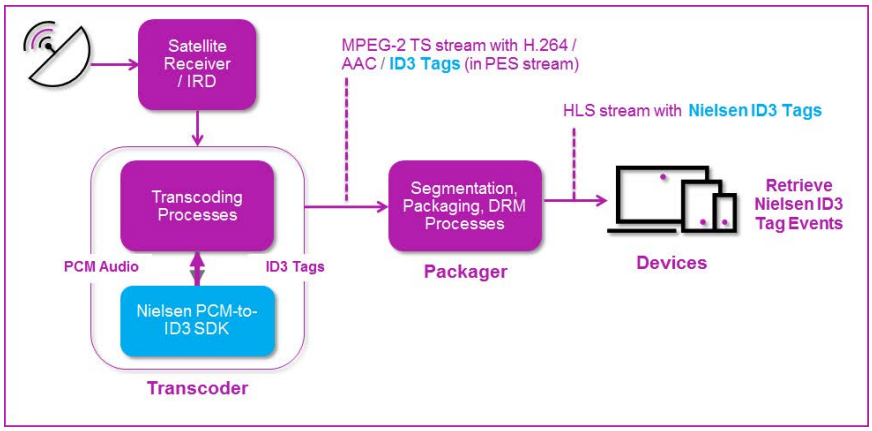

Figure 2 – PCM-to-ID3 SDK Embedded in a Transcoder

2.3. Tasks Performed by Your Transcoder

To serve as a Nielsen ID3-tag generator, your application must perform these basic tasks:

- Extract Nielsen-watermarked audio from a "container" stream such as MPEG-2 or MPEG-4
- Convert the extracted stream to single-channel PCM audio. See section 4 "Audio Input."
- Deliver the single-channel audio to the Nielsen PCM-to-ID3 SDK software, which returns Nielsen ID3 tags
- Remove from the original "container" stream any pre-existing Nielsen ID3 tags
- Insert into the output package (MPEG-2 transport stream, MPEG-DASH stream, or HLS stream) the newly generated ID3 tags. Your application may also edit or transcode the audio and video elementary streams.

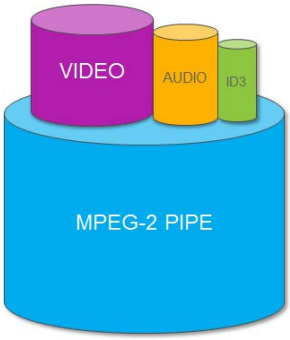

Figure 3 – ID3 Data in an MPEG-2 Transport Stream

2.4. Tasks Performed by the Nielsen PCM-to-ID3 SDK

The PCM-to-ID3 SDK accepts a single channel of PCM audio sampled at one of the supported frequencies defined in section 4 “Audio Input”. As shown in Figure 4, the SDK down-samples the input audio to a 24-KHz sample rate, extracts audio watermarks, and creates ID3 tags. It delivers the ID3 tags to your application.

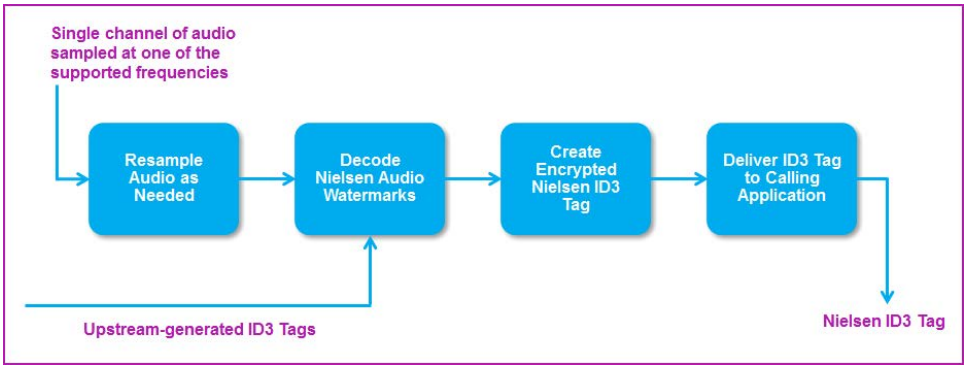

Figure 4 – Processing Flow within the Nielsen SDK

2.5. Quick Start

Now, let’s put all the pieces together to define ID3-tag generation within your application, start to finish. For the purpose of this discussion, let’s assume that you are processing an MPEG-2 transport stream.

1. Your application or system identifies the primary audio elementary stream as the only audio stream that you deliver to the PCM-to-ID3 SDK. We define the primary stream as the first audio Packetized Elementary Stream (PES) listed in the Program Map Table (PMT) of the transport stream.
2. Your application creates a single monaural PCM audio stream from the primary audio PES. For detailed instructions about the generation of a monaural PCM audio stream, see section 4.3 “Audio from Stereo Source” and section 4.4 “Audio from Multi-Channel Source.”
3. Your application calls the API method `InputAudioData` to deliver the single-channel audio (in chunks no greater than 1 second of playback time) to the Nielsen PCM-to-ID3 SDK.
4. The SDK generates Nielsen ID3 tags based on codes embedded in the incoming audio stream. It delivers each ID3 tag to your application using a callback. If your application is generating an MPEG-DASH stream, proceed to step 7. If it is generating an HLS stream, continue to step 5.
    - The SDK delivers DATA tags at a frequency of one ID3 tag every 5 to 10 seconds.
    - The SDK delivers one INFO tag every five minutes.

5. If your application is generating an MPEG-2 transport stream, it embeds the newly generated ID3 tag into a properly formed PES packet, multiplexed into the outbound MPEG-2 transport stream. The multiplexing process must follow the guidelines for inserting ID3 tags into a transport stream specified by Section 2 of *Timed Metadata for HTTP Live Streaming*. See section 1.4, “References and Related Documents.”

   Specifically, your application must:

   - Insert a 17-byte metadata pointer descriptor into the program-info loop of the PMT. See Table 1 “Metadata Pointer Descriptor” in section 2.6.59 of *ISO/IEC 13818-1:2007.*
   - Add the following entry to the elementary-stream loop of the PMT:
     - Stream type = 0x15 (decimal 21)
     - Elementary PID = PID of the ID3 tag stream
     - ES Info Length = 15
     - Metadata descriptor: 15-byte descriptor, as described in “Timed Metadata for HTTP Live Streaming”

6. If you are generating an HLS stream, after completing step 5, your application (or a downstream application) generates a multi-profile HLS package based on the transport stream created in step 5.

7. If your application is generating an MPEG-DASH stream, it inserts the ID3 tag into the Event message (emsg) box associated with the audio segment to which the ID3 tag applies. See “Appendix D—Insert ID3 Tags into an MPEG-DASH Stream” for detailed instructions.

2.6. Deprecated Streaming Modes and Functions

For a brief period between PCM-to-ID3 SDK version 2.1 and 2.3, there were three supported streaming modes. Streaming mode 0 (`PCMID3_STREAMING_MODE_LIVE_STREAMING_CONTENT`) is now the ONLY supported stream mode.

Your application must NOT call the following, deprecated `CPcmToId3Api` methods, most of which are related to the unsupported streaming modes. In version 2.4 of PCM-to-ID3 SDK, these functions have all been removed from the SDK interface:

- `InitializePreParser()`
- `PreParseBuffer()`
- `Report()`
- `GetMetadataInformation()`
- `InputId3PassThroughData()`

2.7. Removing Pre-existing Nielsen ID3 Tags

If the incoming transport stream contains Nielsen ID3 tags, those tags must NOT be included in the output of your application.

In a transport stream or HLS stream, Nielsen ID3 tag elementary stream(s) may be identified as follows:

1. In its elementary stream loop, the PMT of a program with ID3 tags includes an entry identifying an ID3 tag metadata stream. The stream type of that entry is 0x15, and an ID3 tag metadata descriptor follows the entry. Save that PID for use in step 2.
2. From the PES identified by the PID(s) stored in step 1, extract and parse the introductory bytes of the payload of a complete PES packet. The payload of a Nielsen ID3 tag PES begins with a 10-byte ID3 tag header, followed by a 10-byte ID3 frame header. Immediately after the frame header, the Owner ID field begins with the string *www.nielsen.com*. This string positively identifies the PES as a Nielsen ID3 tag PES.

In an MPEG-DASH package, Nielsen ID3 Tags are contained in emsg boxes identified by the scheme ID URI `www.nielsen.com:id3:v1" value="1"`.

If your application identifies any pre-existing Nielsen ID3 tags in the content, those tags must **not** be included in the stream that your application generates.

3. Nielsen ID3 Tags

3.1. Tag Types

There are two different types of ID3 tags: INFO Tags and DATA Tags. For a detailed description of these two types, see the *Nielsen ID3 Tag Validator Application User Guide*.

3.1.1. INFO Tag

The INFO tag holds information about the device that generated the ID3 tags, including definitions of the device type, device identifier, and software version. It also provides information about the SDK and content distributor. This information may be used to identify and troubleshoot problems in the field.

Important **Your application must provide a way for users to configure the Distributor ID.**

For instructions on setting the INFO-tag fields, see section 5.5 “CPcmToId3Properties Object.”

3.1.2. DATA Tag

The DATA tag holds information derived from Nielsen watermarks detected in the audio stream. The watermark information is stored in *EDUs* (Elemental Data Units). A single ID3 tag may hold between 1 and 10 filled EDUs, each representing a single watermark. The SDK does not generate an empty tag.

The EDUs are based on two primary types of watermarks: Program Content (PC) and Final Distributor (FD). The PC watermarks are used, as the name implies, to track program content. The FD watermarks are used to track the distributor of the content.

In addition to the EDUs, each DATA tag includes these fields:

- **Sequence number:** a 16-bit value that is incremented after each ID3 tag is finalized and sent to your application. The value rolls over to 0 after 65,535. If you check the tag sequence numbers, you can determine whether any ID3 tags have been dropped or duplicated during the process of multiplexing or packaging.
- **PC and FD CIDs (Content IDs):** each of the two CIDs is a 24-byte encrypted string that (prior to encryption) has two components, both based on the EDUs within the ID3 tag:
  - **CID SID:** the SID (PC or FD) that best represents the EDUs in the ID3 tag
  - **CID Timestamp:** a 32-bit value representing the start of the most recent broadcast day, where the broadcast day is defined as starting at 3 a.m.
- **PC and FD Offsets:** each offset represents the number of seconds (within roughly 10 seconds) that watermarks in the tag are offset from 3 a.m.
- **Tag times (first and last):** The first tag-time represents the time when data collection

began for the current tag. The last tag-time represents the time when the tag was generated and sent to your application. The tag times are based on a virtual clock that your application exposes to the PCM-to-ID3 SDK. To set the tag times, the SDK uses the third argument that your application provides to the function `InputAudioData()`.

- **Breakout Code:** The breakout code is no longer used; the SDK sets it unconditionally to 0. The Breakout Code Get/Set functions are still defined in the property header file (for backward compatibility), but the functions do nothing.

3.2. Tag Size

Each INFO tag and each DATA tag is 271 bytes. Because the tag exceeds the 188-byte size of a transport packet, it must span two transport packets.

3.3. Tag Timing

A single INFO tag is generated once every five minutes. Either of two events triggers the generation of a DATA tag:

- Ten seconds have elapsed since the last DATA tag was created and sent to your application.
- The EDU array (with 10 slots) was filled before ten seconds elapsed.

Note that the timing of both the INFO and the DATA tags is driven entirely by the `currentTime` argument that you feed to the `InputAudioData()` function.

3.4. Tag Synchronization

It is extremely important that each ID3 DATA tag be synchronized within 10 – 15 seconds with the audio that it represents. To achieve this synchronization, make certain to set the PTS of the ID3 Tag PES packet to the PTS of the last audio PES that your application processed. If your application generates an MPEG-DASH stream, the emsg that holds the ID3 tag must be included in the same segment as the audio block that was most recently processed.

4. Audio Input

4.1. Overview

If the incoming transport stream contains more than one audio stream for a single program, you must present ONLY the PRIMARY audio stream to the SDK.

Important The incoming audio must be "clean," that is, the audio must be relatively free of noise and have no aliasing or distortion. There must be no clipping between 750 Hz and 6.3 KHz. You must make certain that any resampling of the audio does not significantly degrade the audio quality.

The SDK supports uncompressed (PCM) audio streams of 24-bit or 16-bit resolution with the following sample rates:

- **24 kHz:** this is the preferred sample rate, as samples at 24 kHz do not require additional processing for sample rate conversion.
- **48 kHz, 44.1 kHz, 32 kHz, 22.05 kHz, 16 kHz:** If you provide audio at any of these frequencies, the PCM-to-ID3 SDK re-samples the audio to 24 kHz.

4.2. 16-, 24- and 32-bit Sample Size Audio Processing

The SDK processes only PCM audio of 16- and 24-bit sample sizes; however, the 24-bit audio may be packed in a 32-bit container, which requires a padding byte to be placed before or after the 24 bits of actual data. Below are the four possible layouts. All layouts assume the sample is in little endian byte order. For more information regarding these layouts, see the `SetSampleSizeAndPackingMode()` method in section 5.5 "CPcmToId3Properties Object."

1. Figure 5 shows the layout for 16-bit audio packed in 2 bytes. No padding is required because the SDK does not require padding instructions.

Figure 5 – 16-Bit Audio Packed in 2 Bytes

2. Figure 6 shows the layout for 24-bit audio packed in 3 bytes. No padding is required because the SDK does not require padding instructions.

Figure 6 – 24-Bit Audio Packed in 3 Bytes

3. Figure 7 shows the layout for 24-bit audio packed in 4 bytes with MSB padding. The padding byte is the most significant byte of the 32-bit sample. Order is assumed little-endian byte.

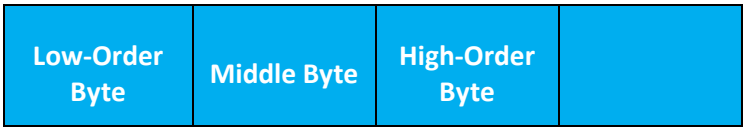

Figure 7 – 24-Bit Audio Packed in 4 Bytes with MSB Padding

4. Figure 8 shows the layout for 24-bit audio packed in 4 bytes with LSB padding. The padding byte is the least significant byte of the 32-bit sample. Order is assumed little-endian byte.

Figure 8 – 24-Bit Audio Packed in 4 Bytes with LSB Padding

4.3. Audio from Stereo Source

As the SDK accepts only one monaural audio channel, if the incoming audio is stereo, your application must pass only the left audio channel to the SDK. **Do not down-mix left and right channel audio into a single monaural mix for the audio input to the SDK**.

Figure 9 – Processing Audio Derived from Stereo Source

4.4. Audio from Multi-Channel Source

For multi-channel audio (5.1), the center channel audio is to be attenuated -3 dB and added to audio from the left channel. Your host application is responsible for combining these channels. As shown in Figure 10, you must scale the audio prior to mixing to prevent overflow of PCM sample values. Nielsen recommends scaling individual audio channels down by -3 dB prior to the mixing process.

Optionally, you may use a Dolby® ProLogic® II down mix (Lt or Lo) instead of applying the above technique. Once you have combined the audio channels, you may process the audio as recommended in section 4.3, “Audio from Stereo Source.”

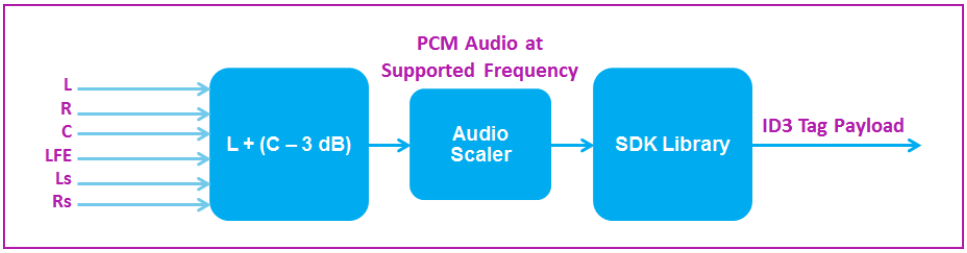

Figure 10 – Preprocessing Required for Multi-Channel Audio

5. SDK Package

5.1. SDK Package Description

In the root folder, the SDK package contains the PCM-to-ID3 SDK libraries, header files, and sample application (including source code). It also holds compiled, executable versions of the sample application for each supported platform. You may select from a set of zipped files, each holding components required for a specific platform, as described in Table 1. Version 2.4 of PCM-to-ID3 SDK supports only 64-bit platforms.

Table 1 – Contents of SDK Package

Platform	Folder	Contents
All	README	Legal information
All	\docs	PCM-to-ID3_SDK_Manual.pdf— this document Release notes (optional)
All	\apps	An executable compiled from the sample application source code, customized for the platform (Linux, Windows, Mac) that you have selected.
All	\lib	The lib directory holds the libraries to which your application must link. For Mac and Linux, the set of libraries are: libCryptopp.a libId3TagUtils.a libNaes2HybridDecoder.a libNaes6Decoder.a libNielsenAudioCore.a libPcmToId3Sdk.a If you are building on a Windows computer using Visual Studio, all of the component libraries are combined in libPcmToId3Sdk.lib.

Platform	Folder	Contents
All	\inc	C/C++ header files required to interface with the SDK PcmToId3Api.h - Defines the primary interface to the PCM-to-ID3 SDK libraries. IPcmToId3Callback.h - Defines the abstract base class from which you must derive your callback class. PcmToId3Properties.h—Defines a class that manages the configurable properties that must be passed to the API. SdkTypes.h—types used by the APIs IPcmToId3Api.h – A base class required by PcmToId3Api.h. Your application will not interface directly with this file.
Linux	\sample	Makefile or Visual Studio project/solution files required to build the sample application.
All	\sample\App	PcmToId3App.cpp—main entry point of the sample application PcmToId3Callback (.cpp and .h) — callback class that receives ID3 tags and log messages from the API. PcmToId3Processor (.cpp and .h)—parses, validates, and processes parameters from the user. Configures and runs the PCM-to-ID3 engine.
All	\Common	Holds source code responsible for reading input files and for writing output files.

5.2. SDK Static Libraries

You may build your application on any of the following platforms:

- Microsoft Windows 7 (64-bit) or later operating system

  You may compile the sample application with Visual Studio 2013 or 2015 (both in the same package). For each supported version of Visual Studio, there are two separate sets of libraries: one built with statically linked C-runtime, and the other built with dynamically linked C-runtime.

- Linux 64-bits (Nielsen built the package on a Ubuntu 16.04 system)

  Nielsen compiled the sample application with gcc/g++ (version 5.4.0). The Linux libraries are compiled with the -fPIC option, allowing you to link to executable files, static libraries, or shared libraries.

- Mac® (Tested using MacOS X Sierra). Use gcc/g++ Version 5.4.0 or later.

5.3. Using the SDK

Below are the objects that the sample application uses to demonstrate the functionality of the SDK. Note that the source code is provided solely as an example. You should customize the software to meet the needs of your application. You are responsible for adding the error-handling capabilities.

- `IPcmToId3Callback`: ID3-tag and log callback interface from which you must derive your own class
- `CPcmToId3Callback`: class you derive from `IPcmToId3Callback` to handle three items that the SDK delivers to you:
  - Log messages: you must store these in whatever logging scheme you support.
  - ID3 tags: you must insert these into the output transport/HLS stream or MPEG-DASH stream.
  - NAES 2 and NAES 6 JSON strings: You may capture these for debug purposes only. This option should be turned OFF for production-ready applications.
- `CPcmToId3Api`: primary API through which your application communicates with the Nielsen SDK libraries
- `CPcmToId3Properties`: class your application uses to set all properties that the SDK requires

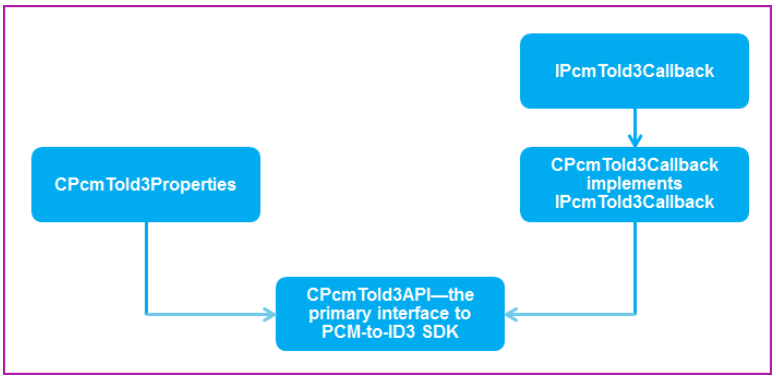

Figure 11 – PCM-to-ID3 SDK Layout

5.4. IPcmToId3Callback.h Interface

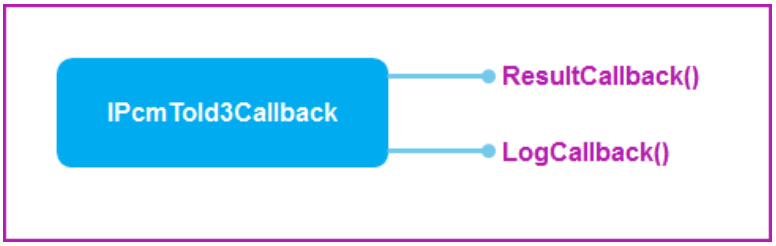

Figure 12 – IPcmToId3Callback Method Calls

`IPcmToId3Callback.h` defines the abstract base for a callback class that your application must create. `CPcmToId3Api` uses this callback class to deliver ID3 tags (`ResultCallback`) and log messages (`LogCallback`) to your application. Note that a third callback (`WatermarkCallback`) is provided for test/debug purposes, to report NAES 2 and NAES 6 codes that the Nielsen decoder extracts from the audio stream. You must construct `CPcmToId3Api` with a pointer to your callback class.

In the Nielsen sample application, `CPcmToId3Callback` is a simple example of a derived callback class. As required, `CPcmToId3Callback` provides implementations for the two pure-virtual functions, `ResultCallback()` and `LogCallback()`. You must replace `CPcmToId3Callback` with a similar class – one that handles `ResultCallback()` and `LogCallback()` in a way that is appropriate for your application.

Note The `WatermarkCallback()` function is provided only for test/debug purposes. It returns a JSON string that holds the settings of each extracted audio code.

```
virtual void ResultCallback(uint8_t *pBuff, uint32_t
len)
```

The SDK library invokes a call to `ResultCallback()` when it has an ID3 tag to deliver to your application.

Note the following regarding `ResultCallback()`:

- Normally the Nielsen libraries call `ResultCallback()` at intervals that range from 5 to 10 seconds (to report an ID3 DATA Tag).
- If there are no watermarks in the audio content, the libraries do NOT generate DATA tags during the period in question. For watermark-free audio content, the SDK libraries generate only INFO tags, and those occur at five-minute intervals.
- Your application must copy all <len> bytes from address <pBuff> into the HLS or MPEG-DASH container that holds the ID3 tag. It must not truncate or modify the ID3 tag in any way (except to split it, as required, between transport packets).
- PCM-to-ID3 SDK owns <pBuff> and is responsible for allocating and freeing memory for the buffer. Your application should neither allocate nor delete this memory.

```
virtual void LogCallback(uint16_t code, char *pMessage
= 0, uint32_t size = 0 )
```

`CPcmToId3TagApi` calls `LogCallback()` when it has errors or status messages to report. If the size argument is set to 0, then you may assume that the `pMessage` argument (if not null) is a null-terminated string. Nielsen requires that you record in your log file all messages delivered by `LogCallback`.

5.5. CPcmToId3Properties Object

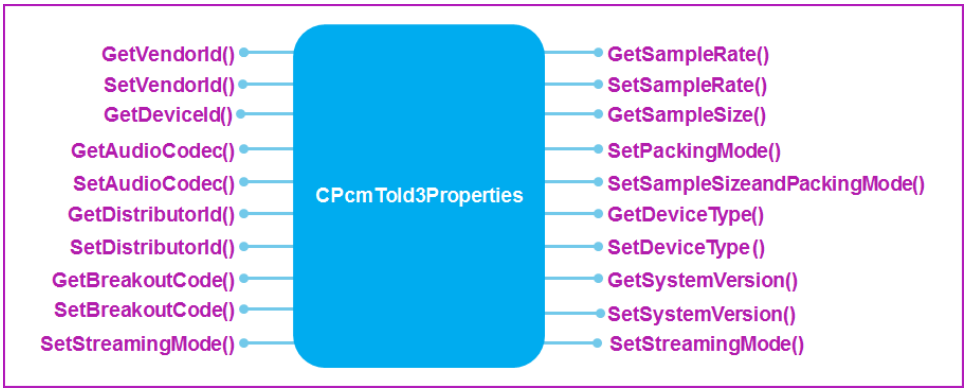

Figure 13 – CPcmToId3Properties Method Calls

The `CPcmToId3Properties` class allows you to set all fields of the INFO tag. When you have properly configured an object of type `CPcmToId3Properties`, you must deliver a pointer to the object to the `CPcmToId3Api` constructor.

Note that, for the purpose of backward compatibility, we have not removed the Get/Set methods for breakout-code and streaming-mode. However, the SDK hard-codes the settings for both of these fields to 0, ignoring any “Set” calls that your sample application may invoke.

```
PCMID3_SAMPLE_RATE_E GetSampleRate()
bool SetSampleRate(PCMID3_SAMPLE_RATE_E sampleRate)
```

`GetSampleRate()` and `SetSampleRate()`: number of samples per second per channel of the source PCM audio stream. The supported rates in Hz are 48000, 44100, 32000, 24000, 22050, and 16000. If your application indicates that the audio frequency is any value other than 24 kHz, the Nielsen software resamples the audio to 24 kHz.

```
bool SetSampleSizeAndPackingMode(PCMID3_SAMPLE_SIZE_E
sampleSize, PackingMode_t mode)
```

`SetSampleSizeAndPackingMode()`: sets the packing mode and number of bits in each audio sample (16 or 24). If the sample size and packing mode combination is not valid, this method returns an error.

Supported sample size and packing mode combinations are:

- For 16-bit samples, packed in a 16-bit container:

  Set sampleSize to PCMID3_SAMPLE_SIZE_16BITS

Set mode to PACKING_MODE_NOT_USED.

- For 24-bit samples, packed in a 24-bit container:

  Set sampleSize to PCMID3-SAMPLE_SIZE_24BITS

  Set mode to THREE_BYTES_LITTLE_ENDIAN

- For 24-bit samples, packed in a 32-bit container, where the least significant byte is the padding byte:

  Set sampleSize to PCMID3-SAMPLE_SIZE_32BITS

  Set mode to FOUR_BYTES_LITTLE_ENDIAN_0_24_PACKING

- For 24-bit samples, packed in a 32 bit containing, where the most significant byte is the padding byte:

  Set sampleSize to PCMID3-SAMPLE_SIZE_32BITS

  Set mode to FOUR_BYTES_LITTLE_ENDIAN

```
PCMID3_SAMPLE_SIZE_E GetSampleSize()
```

`GetSampleSize()`: returns an enum indicating the sample size of the audio.

```
PackingMode_t GetPackingMode()
```

`GetPackingMode()`: returns the packing mode. If each sample is 24 bits in size, the packing mode defines how the three bytes of audio are packed in one of the follow container types:

- In a 3-byte container, no padding required:

  THREE_BYTES_LITTLE_ENDIAN = 0

- In a 4-byte container, where the most significant byte is the padding byte:

  FOUR_BYTES_LITTLE_ENDIAN = 1

- In a 4-byte container, where the least significant byte is the padding byte:

  FOUR_BYTES_LITTLE_ENDIAN_0_24_PACKING = 2

For a more detailed explanation, see section 4.2 “16-, 24- and 32-bit Sample Size Audio Processing.”

```
PCMID3_DEVICE_TYPE_E GetDeviceType()
bool SetDeviceType(PCMID3_DEVICE_TYPE_E type)
```

`GetDeviceType()` and `SetDeviceType()`: get/set methods for the type of device on which the application is running. The follow values are supported:

- PCMID3_RESERVE_DEV = 0 (reserved)
- PCMID3_STREAMING_DEV = 1 (in-home streaming devices)
- PCMID3_TRANSCODER_DEV = 2 (transcoder - this is the most common)
- PCMID3_SEGMENTER_DEV = 3 (HLS segmenter)

```
void GetSystemVersion(uint8_t *pszSystemVersion, size_t
size)
void SetSystemVersion(uint8_t *pszSystemVersion, size_t
len )
```

`GetSystemVersion()` and `SetSystemVersion()`: Get/Set methods for the INFO tag system-version field. You have only 4 bytes plus a null-terminator to represent your system version. If that size is insufficient to hold the entire version number, then set the INFO-tag field to a value that uniquely identifies the actual system version of your product.

For example, assume that the version string for your released product is 5.2.3.401 and that your next release will be 5.2.4.102. You could use “0001” to represent 5.2.3.401 and “0002” to represent 5.2.4.102. The string setting is your choice. However, there must be a way to link it to a specific, complete system version number upon request.

If your system-version string is shorter than 4 characters, simply null-terminate the string after the last character. The SDK will fill the remainder of the string with spaces.

The `size` argument of the Get function defines the number of bytes that you have allocated to the receiving buffer. The <len> argument of the Set function is optional. You may simply null-terminate `pszSystemVersion` and present it as the only argument to the Set function.

```
void GetVendorId(uint8_t *pszVendorId, size_t size)
void SetVendorId(uint8_t *pszVendorId, size_t len)
```

`GetVendorId()` and `SetVendorId()`: Get/Set methods for the vendor ID field of the INFO tag. Nielsen assigns a 3-digit vendor ID to you before you certify your transcoder equipment.

The vendor ID is represented as a 3-byte character string, followed by a null terminator.

The `size` argument of the Get function defines the number of bytes that you have allocated to the receiving buffer. The `len` argument of the Set function is optional. You may simply null-terminate pszVendorId and present it as the only argument to the Set function.

```
void GetDeviceId(uint8_t *pszDeviceId, size_t size)
void SetDeviceId(uint8_t *pszDeviceId, size_t len )
```

`GetDeviceId()` and `SetDeviceId()`: Get/Set methods for the device ID field of the INFO tag. The device ID identifies the device hosting the server software. The ID may be up to 17 bytes long (16 bytes of data, plus a null terminator byte). If you present a string that is too long, the Nielsen software truncates the character string after 16 bytes, setting the last byte to a null-terminator.

The `size` argument of the Get function defines the number of bytes that you have allocated to the receiving buffer. The `len` argument of the Set function is optional. You may simply null-terminate `pszDeviceId` and present it as the only argument to the Set function.

```
void GetAudioCodec(uint8_t *pszAudioCodec, uint32_t
size)
void SetAudioCodec(uint8_t *pszAudioCodec, uint32_t
size )
```

The audio_codec field is no longer used. You may leave it empty.

`GetAudioCodec()` and `SetAudioCodec()`: Get/Set methods for the audio-CODEC INFO tag field. The original intent of the audio-codec field was to represent the CODEC type, the number of channels, and the bit rate of the audio such as "`AC3 2.0 192kbps`". The Audio Codec field is 16 bytes long plus a trailing null-terminator.

The `size` argument of the Get function defines the number of bytes that you have allocated to the receiving buffer. The `len` argument of the Set function is optional. You may simply null-terminate pszAudioCodec and present it as the only argument to the Set function.

```
void GetDistributorId(uint8_t *pszDistributorId, size_t
size)
void SetDistributorId(uint8_t *pszDistributorId, size_t
len)
```

`GetDistributorId()` and `SetDistributorId()`: Get/Set methods for the INFO Tag Distributor ID field.

The distributor ID, a 48-character field followed by a null-terminator, identifies the distributor of the HLS or MPEG-DASH stream, such as a local broadcast affiliate, MVPD, or CDN. In most cases, you must provide a method for the end user of your equipment to set the value of the string. For example, they could set the distributor ID to the URL of the end user's website. During the certification process, you may use a default value representing your own company ("www.xyz.com"), but you are required to show that your application or system provides a method for the end-user to reset the value of the string.

If you propose a setting that is too long, the Nielsen software truncates your string after the 48th character.

The `size` argument of the Get function defines the number of bytes that you have allocated to the receiving buffer. The `len` argument of the Set function is optional. You may simply null-terminate `pszDistributorId` and present it as the only argument to the Set function.

5.6. CPcmToId3Api Object

`CPcmToId3Api` is your primary interface to the PCM-to-ID3 SDK libraries. Its constructor accepts a pointer to the **CPcmToId3Properties** object and a pointer to the **CPcmToId3Callback** object. See section 5.5 "CPcmToId3Properties Object."

Figure 14 – CPcmToId3Api

```
virtual uint16_t Valid()
```

`Valid()` returns 0 if and only if the class constructor executed without error.

If an error was encountered, the `Valid()` function returns a negative error code.

```
virtual bool Initialize()
```

`Initialize()` configures the application prior to the start of audio processing.

If the function runs to completion, it delivers one or more log messages that indicate the version number of PCM-to-ID3 SDK and of one or more of its component libraries. **Be sure to store the log message on your system so that users can retrieve them for diagnostic purposes if they encounter problems with the transcoder.**

```
void Release()
```

`Release()` releases system resources.

```
virtual bool GetEngineVersion(char *pName, uint32_t
size)
```

`GetEngineVersion()` sets <pName> to a string that provides the software version numbers of PCM-to-ID3 SDK and one or more of its component libraries. The size argument indicates the number of bytes that you have allocated to pName. **Be sure to allocate at least 80 bytes to the pName buffer**.

The function returns true if and only if successful.

```
virtual void InputAudioData(uint8_t *inBuffer, uint32_t
nSize, time_t currentTime)
```

`InputAudioData()` accepts PCM audio data from your application and submits the data to the PCM-to-ID3 engine. It is a blocking call that does not return until the entire buffer has been processed.

- `inBuffer` points to the audio data to be processed
- `nSize` specifies the number of bytes of data in `inBuffer`.

- `currentTime` is a value that must increase by 1 for each second of processed data. For streaming applications, you may use the POSIX time code synchronized to the system clock. If the source audio content is read from file, however, `currentTime` must accurately reflect the number of seconds of audio data that have been processed since the beginning of this ID3-tagging session. Note that if you are generating an MPEG-DASH package, you could use the `currentTime` setting to populate the id field of the emsg box.

While executing the `InputAudioData()` function, the SDK library may invoke a call to `ResultCallback()` in order to deliver a Nielsen ID3 tag to your application. Most of your calls to `InputAudioData()`, however, will result in no calls to `ResultCallback()` at all, since DATA tags occur at 5-to-10 second intervals, and INFO tags occur only once every 5 minutes.

It is also possible that `InputAudioData()` will invoke one or more calls to `LogCallback()` to deliver a status or error message to your application. You should include Nielsen log messages in your log file.

5.7. SdkTypes.h

- `SdkTypes.h`
  - Describes some of the basic types used by the SDK
  - Validates some of the configuration properties
- `PcmToId3Types.h` – This header file was included in earlier versions of the SDK, but is not included in version 2.4. **Where applicable, replace #include "PcmToId3Types.h" with #include "PcmToId3Properties.h".**
- `w32stdint.h` This file is no longer included in the set of header files. All modern compilers now recognize <stdint.h>. **Replace all instances of #include "w32stdint.h" with #include <stdint.h>.**

6. SDK Sample Application

The sample application demonstrates how to use the SDK, but does not exercise all of the SDK capabilities. The sample application assumes that the incoming PCM audio stream is stored as a WAV file. However, your application most likely will pass in raw PCM audio, not in WAV format. The simplified WAV reader is provided only for demonstration purposes.

If you are upgrading an existing application to PCM-to-ID3 SDK, version 2.4, please remember that the new SDK hard-codes the streaming mode to 0 and hard-codes the breakout-code to 0, ignoring any “Set” calls that your application invokes for either of these two fields.

6.1. Parameters Usage

The sample application creates a binary file holding the generated Nielsen ID3 tags. The name of the tag file is:

```
<original WAV file name>_id3tag.out
```

On loading the WAV file, the SDK checks whether the WAV file has invalid parameters (for example, an invalid sample rate or sample size). If the SDK finds an invalid parameter, it stops processing and displays an error report on the console. To use the sample application, call it from a prompt as follows:

```
PcmToId3 -i <file> [-o <outdir>][-l <logdir>]
```

Where:

- `-i <file>` is the full path and file name of the WAV file to process. PcmToId3 accepts a WAV file as input, extracts all PCM information from its header, detects Nielsen watermarks, and generates a file holding generated ID3 tags.
- `-o <outdir>` is the full path name of the folder that holds the generated Nielsen ID3 tags. This is an optional parameter; if it is not provided, the output file is written to the same location where the original source file is located.
- `-l <logdir>` is the full path of the folder that holds error and status reports. This is an optional parameter. If it is not provided, the log file is written to the same location where the original input file is located.
- `-pm <mode>` is the packing mode, which is required to process audio with 24-bit samples. When processing WAV files whose audio has 24-bit (or 24-bit packed as 32-bit) samples, you must specify the packing mode. For audio with 16-bit samples, do not use the `-pm` option.

  0 = THREE_BYTES_LITTLE_ENDIAN

  1 = FOUR_BYTES_LITTLE_ENDIAN (24_0 packing)

  2 = FOUR_BYTES_LITTLE_ENDIAN_0_24_PACKING

3 = PACKING_MODE_NOT_USED (default)

See section 6.2, “PcmToId3 Components,” for more detailed information.

- `-h` displays command-line options usage. This is an optional parameter. It is ignored if used with other commands. The usage can also be displayed if the application is called with unexpected parameters or no parameters at all.
- `-v` is an optional parameter that allows you to turn on verbose debug messages (primarily to mark the generation of INFO tag or to turn on the reporting of audio codes through the `WatermarkCallback()` function). This parameter is provided only for the purpose of testing/debugging.
- -loop is an optional parameter that is used only for the purpose of continually reprocessing a single input WAV file. It is provided ONLY for the purpose of testing/debugging. The output ID3 tags that the application generates when run in loop mode will contain many errors in fields that pertain to watermark times (because the times roll backward at the loop-back point).

6.2. PcmToId3 Components

The sample application is comprised of three main elements:

- `PcmToId3App.cpp` file
- `CPcmToId3Callback` (.cpp and .h) object
- `CPcmToId3Processor` (.cpp and .h) object

The sample application must statically link to the `PcmToId3Sdk` library or libraries.

6.2.1. PcmToId3App File

The `PcmToId3App.cpp` file is the main entry point to the sample application. It instantiates the `CPcmToId3Processor` object which, in turn, parses and interprets the command-line parameters.

6.2.2. CPcmToId3Callback (.cpp and .h) Object

The `CPcmToId3Callback` object implements the `IPcmToId3Callback` interface that defines the callback methods. Before creating the `CPcmToId3Api` object, your application must create an instance of your callback class and then pass a pointer to the callback object to `CPcmToId3Api`.

This callback object implements the two of the pure-virtual methods defined in `IPcmToId3Callback`:

```
virtual void ResultCallback(uint8_t *pBuff, uint32_t
len)
```

`CPcmToId3Api calls ResultCallback()` whenever it has an ID3 tag to deliver. .

```
virtual void LogCallback(uint16_t code, char *pMessage
= 0, uint32_t size = 0 )
```

`CPcmToId3Api` calls `LogCallback()` whenever it has an error or status message to report.

At roughly 10-second intervals (or less), the `CPcmToId3Api` object invokes the `ResultCallback()` method to deliver DATA tags to your application. `CPcmToId3Api` generates INFO tags at 5-minute intervals. The length of these intervals is approximate.

In the SDK sample application, the DATA and INFO tags are stored in a file. In contrast, your application must copy the entire unaltered 271-byte ID3 tag into the output (HLS or MPEG-DASH) stream.

6.2.3. CPcmToId3Processor (.cpp and .h) Object

The `CPcmToId3Processor` object instantiates and configures the `CPcmTpId3Callback` and `CPcmToId3Properties` objects. It instantiates and directly interfaces with the SDK through the `CPcmToId3Api` class. It parses, validates, and processes all parameters passed by the user.

The `CPcmToId3Processor` class is instantiated and destroyed by `PcmToId3App`, and it lasts for the duration of the sample application.

7. SDK Test Files

When you receive your SDK package, Nielsen grants you access to a portal from which you may download the following:

- Three watermarked transport-stream test files
- Nielsen ID3 Tag Validator (a Windows console application that you may use as a diagnostic tool)
- Instructions for certifying your application or device

If you need further support in downloading these files, contact your Nielsen technical representative.

Appendix A—Frequently Asked Questions

1. Can input streams have more than one audio channel?

   The SDK can process only a single audio channel. This document describes the kind of audio pre-processing required for stereo or 5.1 channel audio inputs. For more information, see section 2 “Concept” and section 4 “Audio Input.”

2. Does audio pre-processing occur inside the SDK?

   Your application is responsible for pre-processing the audio (creating a monaural stream); however, in some cases the SDK will resample the audio that you present. The sample application included in this package demonstrates the case of a stereo audio source, where it passes only the left channel of audio to the SDK. For audio in the 5.1 surround format, see section 2, “Concept” and section 4 “Audio Input.”

3. What are the current versions of the SDK and Audio SDK in use?

   To print the current version of the SDK and of the Audio SDK that is contained within it, call the `CPcmToId3Callback::GetEngineVersion()` method. See section 5.5 “CPcmToId3Properties Object.”

4. Is there any tool to decode an *.id3tag.out file generated by the SDK?

   You may download Nielsen ID3 Tag Validator (a Windows console application that you may use as a diagnostic tool). Contact your Nielsen technical representative for more information about the Nielsen ID3 Tag Validator.

5. What are the valid sample frequencies for the audio provided to the SDK? The valid sample frequencies are:

   - 16 kHz
   - 22.05 kHz
   - 24 kHz
   - 32 kHz
   - 44.1 kHz
   - 48 kHz

   Any other value causes the SDK to return an error. The sample application prints the error message in the log file and standard output. The SDK resamples the audio content to 24 kHz before presenting it to the core engine.

6. What are the valid audio sample sizes that the SDK can process?

The only valid sample sizes are 16 and 24-bits. Conditions may exist, however, where a 32-bit audio sample size can be processed. See section 4.2 “16-, 24- and 32-bit Sample Size Audio Processing.” All supported combinations of sample size and packing mode are described in section 5.5, “CPcmToId3Properties Object.” Unsupported sample resolutions produce error messages.

7. Can the SDK library process 32-bit audio?

   Strictly speaking, it cannot correctly process an audio file of 32-bit samples sizes. However, it can process 24-bit audio samples packed in 32-bit containers. See section 4.2 “16-, 24- and 32-bit Sample Size Audio Processing” and section 5.5, “CPcmToId3Properties Object.”

8. What Microsoft Windows operating system platforms does the SDK support?

   The SDK supports Windows 64-bit operating systems. The SDK should work on Windows 7 and later versions. See section 5.2 “SDK Static Libraries.”

9. What Linux® OS versions does the SDK support? Are there any constraints? The SDK supports Linux 64-bit (GCC must be at least version 5.4). See section 5.2 “SDK Static Libraries.”

10. How do I multiplex the ID3 tags into the original transport stream?

    If you are multiplexing a Nielsen ID3 tag PES into the original transport stream, remember to add the required descriptors in the PMT table. These descriptors are described in by Section 2 of *Timed Metadata for HTTP Live Streaming*. See section 1.4, “References and Related Documents.” Note that other rules apply when you are generating MPEG DASH streams. If you are generating MPEG-DASH content, see Appendix D—Insert ID3 Tags into an MPEG-DASH Stream for instructions.

11. If the incoming content has multiple audio streams, which of the audio streams should I process?

    If the program that you are processing contains more than one elementary audio stream, feed only the PRIMARY audio stream to the SDK. The primary audio stream is defined as the first audio stream listed in the PMT elementary-stream loop.

12. What should I do if the characteristics (sample rate, sample size, packing mode, etc.) of the audio being fed to the SDK change?

    Whenever the basic characteristics of the audio stream change, make certain to alert the SDK to those changes by:

    a. Calling the `Release()` method of the `CPcmToId3Api` object
    b. Updating the properties of the `CPcmToId3Properties` object
    c. Calling the `Initialize()` method of the `CPcmToId3Api` object

    Note that, since the SDK only accepts one audio channel at a time and all the audio channels are managed outside the SDK, it is not necessary to reinitialize the SDK if the characteristics of the audio do not change.

As an example, if the host application must switch from stereo to 5.1 audio, the application pre-processes the audio before delivering it to the SDK. In the end, it delivers just a single channel of PCM audio to the SDK. As long as the sample rate, sample size, and packing mode do not change, the host application does not need to reinitialize the SDK.

13. The INFO tag fields (Device Type, System Version, Vendor Id, Device Id, Audio Codec, and Distributor ID) are required to initialize the SDK. Are there any restrictions pertaining to these settings? Are these fields mandatory?

    Yes, most of these fields are mandatory. Only the Audio CODEC is optional. See Appendix C—PCM-to-ID3 SDK UI Settings.

14. If the audio input does not contain Nielsen watermarks, can the PCM-to-ID3 SDK be configured to generate DATA tags every 10 seconds?

    With the current software, no, you cannot. If non-watermarked audio is fed to the PCM-to-ID3 SDK, no DATA tags are generated.

15. What VOD breakout code mode setting should I use? Is it different for each content/channel? Should it be exposed in the UI? If yes, at what level, per-application or per-content? Could you provide some explanation of the significance of this setting and how it applies to content such as time-shifting, start-over, catch-up, etc.?

    Always set the breakout code to 0.

16. Is it possible that watermarks are not present in audio input during certain periods? If there are no watermarks in the input audio, does the PCM-to-ID3 SDK instance periodically output something? Would the operator be interested in being informed that a particular channel that is configured to detect Nielsen is not generating Nielsen tags?

    Yes, there are instances where Nielsen Watermarks can disappear, but it is unusual. It normally happens in periods of silence. The PCM-to-ID3 SDK does not generate DATA ID3 tags during those silent periods.

17. What value should be passed in for `p_distributor_id` in the TI 6424 DSP library?

    This has the same meaning as `DistributorId` for the non-DSP software. The field should be set to a string that represents the content distributor. See Appendix E—DSP Library Implementation.

18. If an input transport stream contains multi-language audio elementary streams, should watermarks from all audio elementary streams be processed by a single instance of the SDK and the resulting tags put into a single Nielsen ID3 elementary stream in the transport stream output?

    No. For each program in a transport stream, only the PRIMARY audio PES should be processed. This is acceptable because all of the audio PES for a single program are watermarked with the same codes.

19. If the host application is used by third parties to provide video services, would the Distributor ID value identify the third party?

The Distributor ID is used to identify the end-user of the transcoder software (the final distributor of the audio/video content).

20. What do we need to do to certify our application or device for production use?

    A well-defined certification process is documented on the Nielsen Engineering Forum, where you can download test files and diagnostic tools. Contact your Nielsen technical representative for more information.

21. To be able to differentiate Nielsen ID3 PES from other ID3 PES (still image), would the PES be associated with an NMR1 registration descriptor?

    We do not use registration descriptors. We use the industry standard metadata descriptor, as defined by the ID3.org in the ID3 Engine design document.

Appendix B—Factors that Can Affect Decoding

Nielsen Audio Watermarks were designed to survive audio compression when the bitrate is maintained at the recommended level (Table 2).

Code recovery may be affected when the audio is clipped or near silence.

Table 2 – Acceptable Audio Compression rates

Audio Compression	Compression Rate
AC3 Stereo	192 Kbps or higher
AC3 5.1	384 Kbps or higher
Enhanced AC3	192 Kbps
MPEG2 audio	192 Kbps or higher

Appendix C—PCM-to-ID3 SDK UI Settings

Transcoders that implement the PCM-to-ID3 SDK are responsible for setting the breakout code to 0 and for correctly setting the INFO tag fields. The transcoder user interface must provide a means for the end user to set the breakout code and the Distributor ID.

ID3 INFO Tag Fields

Table 3 – ID3 INFO Tag Field

INFO Tag Name	Description	Value	Comment
Device Type	One-byte value identifying the class of your device. If your device does not fall into one of these types, contact Nielsen technical representative about defining a new device type.	`PCMID3_RESERVE_DEV = 0`	Reserved: do not use.
		`PCMID3_STREAMING_DEV = 1`	In-Home Streaming Device (not typical); for gateway devices that transcode MPEG-2 TS streams to HTTP adaptive bit-rate streaming protocols
		`PCMID3_TRANSCODER_DEV = 2`	Transcoder (most common designation)
		`PCMID3_SEGMENTER_DEV = 3`	Segmenter

System Version	Your software version; helps in tracking down and isolating specific problems in the field	If software version is “2.1.3.1”, the string should be:
		`“2131” + ‘\0’`
		Example showing shorter string:
		`“21” + ‘\0’`
			If the string is longer than 4 bytes, the SDK truncates and null-terminates at the fifth character. It is not necessary to pad shorter strings.

INFO Tag Name	Description	Value	Comment
Vendor ID	Nielsen assigns a vendor ID to you as the vendor of the application or system; this setting helps Nielsen identify devices that experience problems in the field.	Example: `"501" +'\0'`	Please ask your Nielsen technical representative to provide you with a vendor ID if you do not already have one.
Device ID	At a minimum, the device ID should identify the product name of your device or application. Alternatively, you could use the hardware serial number to identify the specific device on which your application runs.	Example: `"V12345678A-3" + '\0'`	If the string is longer than 16 characters, the SDK truncates and null-terminates at the 17th character.
Audio Codec	(Optional) Audio codec that your software uses to generate PCM audio	Example of freeform value: `"AC3 5.1 384kbps" + '\0'`	If the string is longer than 16 characters, the SDK truncates and null-terminates at the 17th character.
Distributor ID	DNS domain name of the company or entity operating the device in the field (the company that is distributing the content). Your system must provide the end-user with a way to configure this setting.	Example showing values padded with trailing spaces. Where possible, add descriptive information at the end of the DNS name to help identify the region or location of the transcoding device. Example: `"www.directv.com" + '\0'`	If the string is longer than 48 characters, the SDK truncates and null-terminates it at the 49th character.

Appendix D—Insert ID3 Tags into an MPEG-DASH Stream

If you plan to package the Nielsen ID3 Tags into an MPEG-DASH container, follow the guidelines in in ISO/IEC 23009-1:2014(E). This appendix describes how you must customize your MPEG-DASH container to include Nielsen ID3 tags.

Media Presentation Description (MPD)

Your MPEG-DASH package must contain a “static” Media Presentation Description (MPD), as described in ISO/IEC 23009-1:2014(E), section 5.3.2.1, where a “static” MPD is defined by these properties

- It does not have an @start attribute.
- The @type attribute is set to “static”.
- The Period element is the first in the MPD. In the Period element of that static MPD, you must include this line, which identifies the in-band event stream that holds Nielsen ID3 tags:

```
   <InbandEventStream
schemeIdUri="www.nielsen.com:id3:v1" value="1"/>
```

Event Message (emsg) Box

Your MPEG-DASH package should carry each Nielsen ID3 tag in the Event message (emsg) box of the segment to which it applies. If audio segments are carried separately from video segments, the Nielsen in-band event message should be carried with the audio segment.

The syntax of the emsg box carrying the ID3 tags is defined in section 5.10.3.3.3 of ISO/IEC 23009-1:2014(E). These byte-aligned fields must follow the box type (“emsg”) and length (4 bytes, with recommended setting of 0):

- **Scheme_id_uri (23 bytes):** a null-terminated ASCII string, set to “www.nielsen.com:id3:v1”

  Note that this string matches the stream_id_uri in the MPD; this requirement is specified in section 5.10.3.3.1 of ISO/ID3 23009-1:2014(E).
- **Value (2 bytes):** a null-terminated ASCII string, set to “1”
- **Timescale (4 bytes):** recommended value is 90000, indicating that the presentation time delta is based on a 90 KHz clock (90,000 ticks per second). This field, however, has no functional use and may be set to 0.

- **Presentation_time_delta (4 bytes):** recommended setting is the difference (in clock ticks) between the time when the ID3 tag event is presented and the earliest presentation time in this segment. However, this field has no functional use and may be set to 0.
- **Event duration (4 bytes):** must be set to 0xffff, indicating an unknown duration
- **ID (4 bytes):** Recommended setting is a value that identifies this instance of the message; for example, you could use a “clock” value that advances by 1 for each second of processed audio. However, this field has no functional use and may be set to 0.
- **Message_data (271 bytes):** the complete 271-byte ID3 tag, starting with the ID3 tag header and ending with the ASCII character “A.” Whenever the PCM-to-ID3 SDK `ResultCallback` returns an ID3 tag, you must copy the entire 271-byte tag into the emsg box message-data. See section 19 of this document for information about the result callback. If there are two or more ID3 tags to be inserted into the emsg box, they must be separated by the ASCII EOL character.

See section 5.10.3.3 of ISO/IEC 23009-1:2014(E) for a complete description of Event message boxes. See section 5.10.3.2 for a description of MPD signaling.

Appendix E—DSP Library Implementation

The DSP library is designed to be linked with the Texas Instruments™ TMS320C6424 Fixed-Point Digital Signal Processor. The development environment is TI's Code Composer Studio™ version 3.3 or later.

Library Memory

The size of the memory buffer is 512 bytes for audio when captured at 48KHz (or 256 bytes for 24KHz) and 271 bytes for the ID3 tag buffer.

The following sections of memory are defined and used inside the library. These sections need to be defined in the application's linker file. Mapping to other sections may be possible but performance could suffer. Memory used in the library is static and is identified by the following data_sections:

- .id3MemInt > IRAM //Internal Memory
- .id3MemExt > DDR2 //External Memory

All large buffers are aligned on cache boundaries (128-byte cache line size for L2 cache on C6000).

Library Development

DSP library development is carried out using the TI C6424 evaluation board and associated framework. The development effort is split into the following projects:

- DSP development project

  This project contains all necessary files for initializing the evaluation board for capturing audio, decoding Nielsen watermarks, and generating the Nielsen ID3 tag buffer.

- DSP library project

  This project only contains source files for PCM fixed-point decoder and ID3 tag utilities. This is a subset of the development project and outputs `pcmId3Lib.lib`.

  The DSP library project includes the `pcmId3Lib.lib` and `pcmId3Lib.h` files.

- DSP test project

  This project is the same as the DSP development project with the exception that it integrates pcmId3Lib.lib with the evaluation board initialization source code.

  This project simulates the integrated framework using pcmId3Lib.lib generated from PCM ID3 library project.

Library Testing

DSP library testing is carried out using the TI C6424 evaluation board. The DSP development project is used for tests by using the DSP/BIOS Log utility. This is only for verifying functionality and not to be used for benchmarking performance as this process is unable to run in real time. Full benchmarking and performance must be carried out as described below.

Library Performance Benchmark

As there are many ways to configure a DSP, here are some examples of the type of performance one may expect. Audio, as shown in the example in Figure 15 is supplied as 1024 samples (512 L, 512 R) in a buffer sampled at 48k. This represents a duration of 10.66 milliseconds.

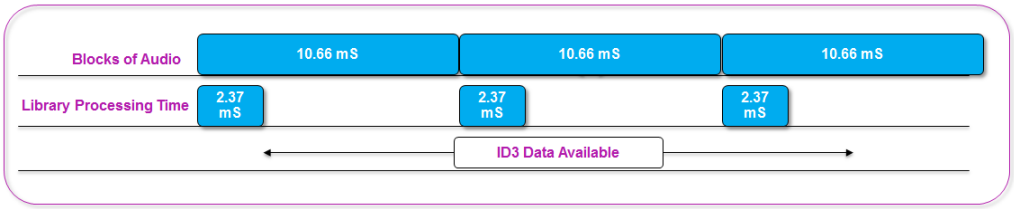

Figure 15 – Timing Diagram for Audio Buffer Processing

With the L2 cache set, program execution takes 2.3 milliseconds when a code is detected.

Audio is accumulated over time in the input buffer. When the audio buffer is full, the function `pcmId3Gen` call is made to the library. In the example above, the buffer size is 256 mono-samples at a 24Khz sample rate. This is the native buffer size for code extraction. The ID3 DATA tag gets generated when all 10 EDU slots are filled or after 10 seconds of elapsed time and at least one tag has been received, whichever occurs first. The ID3 INFO tag gets generated once every 5 minutes.

Library Function Calls

pcmId3LibInit

Use `pcmId3LibInit` to initialize the system as follows:

```
PCMID3_RETURN_E pcmId3LibInit(
uint8_t  *p_vendor_id,
      uint8_t  device_type,
uint8_t  *p_device_id
uint8_t  *p_system_version,
uint8_t  *p_audio_codec,
      uint8_t  *p_distributor_id,
      uint8_t breakout_flag,
PCMID3_SAMPLE_RATE_E audio_sample_rate,
      void *pDecDS,
      void *pId3DS)
```

Table 4 – pcmId3LibInit

Parameters	Description	Value or Example
`vendor_id`	3 bytes, assigned by Nielsen	
`device_type`	Type of the device	0 = Reserved 1 = In-home streaming device 2 = Transcoding device 3 = Stream segmenter/packager 4 – 255 = TBD
`device_id`	16-byte unique ID or serial number	
`system_version`	4-byte version number of the application or system software	
`audio_codec`	Optional, no longer used 16-byte ASCII text field holding the audio codec type description and bitrate Examples: “AC3 5.1,384kbps ” “AAC-HE,192kbps ”	If the string: • Does not fill the 16-character field, you must add trailing spaces. • Is longer than 16 characters, the SDK truncates and null-terminates it at the 49th character. Examples:
`distributor_id`	48-byte MVPD/CDN identifier (distributor)	
`audio_sample_rate`	Sample at which audio will be supplied	0 = 24K 3 = 48K (default is PCMID3_FREQ_48KHZ)

Parameters	Description	Value or Example
breakout_flag	1-byte value	Set to 0
pDecDS	Pointer to watermark decoder context	
pId3DS	pointer to pcmId3 context	

pcmId3Gen

The function prototype is the following:

```
PCMID3_RETURN_E pcmId3Gen(short *pf_pcmBuf, uint32_t
u32_current_time, uint8_t *pu8_id3TagBuf, void
*pDecDS, void *pId3DS);
```

This function call performs all processing and, upon successful generation of an ID3 tag, `pu8_id3TagBuf` is populated with ID3 data. These are the expected return values:

- `PCMID3_RETURN_TAG` indicates successful generation of a tag.
- `PCMID3_RETURN_NO_TAG` indicates no tag has been generated.

Evaluation Board Initialization

C6424 EVM is initialized to capture stereo 1024(512L, 512R) audio sample blocks using AIC 33, McBSP1, and EDMA3 at 48KHz.

Glossary

E

EDU

Elemental Data Unit. A sub-container in the Nielsen ID3 tag that holds an individual Nielsen watermark. A single ID3 tag can hold up to 10 EDUs.

F

FD Watermark

Nielsen Watermark used to track the final distributor (FD) of the audio content, and not the content itself.

H

HLS

HTTP Live Streaming. The Apple-defined HTTP-based adaptive rate streaming protocol.

I

ID3

Apple's solution for delivering metadata in streaming content

ID3 Specification

ID3 refers to a metadata container most often used in conjunction with the MP3 audio file format. The ID3 specification allows information such as the title, artist, album, track number, and other information about a file to be contained within the music file itself. In Nielsen's case, the ID3 tag specification is used to carry Nielsen watermarks.

M

MPEG-DASH

"DASH is an adaptive bitrate streaming technology where a multimedia file is partitioned into one or more segments and delivered to a client using HTTP." (Source: "Dynamic Adaptive Streaming over HTTP." Wikipedia. Accessed 1017-04-04. https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP)

MPD

MPEG-DASH Media Presentation Description. "A media presentation description (MPD) describes segment information (timing, URL, media characteristics like video resolution and bit rates), and can be organized in different ways such as SegmentList, SegmentTemplate, SegmentBase and SegmentTimeline, depending on the use case.[9] Segments can contain any media data, however the specification provides specific guidance and formats for use with two types of containers: ISO base media file format (e.g. MP4 file format) or MPEG-2 Transport Stream." (Source: "Dynamic Adaptive Streaming over HTTP." Wikipedia. Accessed 1017-04-04. https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP)

N

Nielsen Watermark

A proprietary audio algorithm developed by Nielsen to insert inaudible watermark information in the audio portion of media content to enable audience-viewing measurement.

P

PC Watermark

Audio watermark used to track program content (as opposed to the distributor of that content).

PCM

Pulse Code Modulation

PES

Packetized Elementary Stream

PID

Program Identifier

PMT

Program Map Table

S

SID

Source Identifier is Nielsen-specific term. Source IDs are associated with Nielsen watermarks that are injected into ID3 tags by Nielsen's PCM-to-ID3 SDK for retrieval downstream by mobile consumer devices.