SUPPORTING CUE FILES IN MPD: SYNTACTIC AND SEMANTIC CONSISTENCY WITH STANDARD FORMATS FOR AUDIO METADATA

alexus

2019-05-08

Version: 2.0

Copyright (c) 2018+ alexus. Released under the GNU Free Documentation License < https://www.gnu.org/licenses/fdl.html > , Version 1.3 or later.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License < https://www.gnu.org/licenses/fdl.html > , Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

DISCLAIMER: ANY INFORMATION IN THIS DOCUMENT IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED OR STATUTORY WARRANTIES ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

— In memory of my beloved Mother, who pleasantly singing the operas thus led me appreciate all the good music.

1 SUMMARY

Many people prefer to organize their digital audio collection with focus to albums (usually one folder per album), using single lossless FLAC files (one audio file per album) or multiple FLAC files (one file per each track in the album), along with a CUE file playlist containing both the information about the audio file(s) and track durations, and metadata related to the album and its tracks.

Often these are people who own a large physical collection of CDs (and other physical media such as SACDs, LPs, MCs etc.) that they have accurately converted into a digital form, saving them as audio files (WAV, FLAC etc.) both for digital listening and for backup purposes (all the audio supports deteriorate with time and use). Many of them are audiophiles using a hi-fi stereo system (with both analogue and digital components, including a high-quality real-time audio server) for playback listening.

Moreover, like many other people do, they often use the MPD (Music Player Daemon) application server for playback and a MPD client for control.

In such a context, different standard formats for audio are involved (regarding both the sound data and the audio metadata) such as CDDB, TOC/CD-TEXT, CUESHEET, VORBIS COMMENT, MPD TAGS.

Therefore it **is essential that MPD and its clients have the ability to support CUE files in a way that is syntactly and semanticly as consistent as possible with all these standard formats, identifying a common set of audio metadata to be supported and exactly specifying their meaning and use**.

In the following, the involved formats for audio metadata will be analyzed, compared and discussed, to identify a way to properly support CUE files in MPD while also ensuring syntactic and semantic consistency. A reference summary with proposals to standardize and extend both CUE commands and MPD tags and a pseudocode proposal of the implementation, will also be provided.

2 STANDARDS AND FORMATS FOR AUDIO METADATA

Below in this chapter the main and most used standard formats for audio metadata are examined.

2.1 CDDB (XMCD)

CDDB ([1]) was invented in 1993 as a local database delivered with the popular **XMCD** music player; the acronym CDDB (Compact Disc DataBase) was first mentioned in version 1.1 (CDDB1) of the software, released in 1994 under the GPL license < https://www.gnu.org/licenses/gpl.html >. Rapidly the database became very large thanks to user contributions, and in 1995 an online version was created continuing to be increasingly populated by users. Subsequently, the project underwent several evolutions and acquisitions by various companies. Currently there is a proprietary version (CDDB2) of the database, renamed Gracenote, which was acquired by Sony Corp. of America in 2001. The license change motivated many forks in the CDDB project tree, including **freedb** and **MusicBrainz**, which are intended to remain free software under the GPL license < https://www.gnu.org/licenses/gpl.html >.

The need for CDDB was a consequence of the lack of the disc title or track titles in the CDDA ([2]) audio CD format, thus making necessary a database with this information to be used with digital media systems. It is widely used still nowadays, primarily by media players and CD ripping applications which can query CDDB online services to obtaining audio metadata to be transposed to playlists.

2.2 CDDB: FREEDB

freedb < http://www.freedb.org/ > ([3]), released under the GPL license < https://www.gnu.org/licenses/gpl.html >, is one of the most used forks of the original CDDB version (CDDB1). As of May 2019, the database held about roughly 3.88 million CDs.

Each audio CD is identified by a disc ID, calculated with a hash function based on the CD table of contents, and stored in the database among with the artist, the album title, the track list and other some additional information.

The FREEDB format specification ([4]) ([5]) ([6]) includes information on track durations and audio metadata related both to the album and to its tracks.

Entries in the CDDB database must be in the US-ASCII, ISO-8859-1 (ISO-Latin-1) or (as of version 6 of the freedb protocol) in the UTF-8 character set.

The specification provides that the lines at the top of a CDDB file must be *comments* (starting with #) containing general information about the CD, including track durations referred to as timecode frame ([7]) offsets (1 frame = 1/75 second); the following lines in a CDDB file must be *keywords* in “`KEYWORD=data`” format, containing the metadata information about disc (album) and tracks. All of the applicable keywords must be present in the file, though they may have empty data (except for the `DISCID` and `DTITLE` keywords).

Keywords are as follows:

- **`DISCID`**: Calculated DiscID (CDDB1).
  - scope: disc
  - format: characters
- **`DTITLE`**: By convention contains the artist and disc title (in that order) separated by a ‘/’ delimeter with a single space on either side to separate it from the text. If the ‘/’ is absent, it is implied that the artist and disc title are the same, although in this case the name should rather be specified twice, separated by the delimiter. If the disc is a sampler containing titles of various artists, the disc artist should be set to “Various” (without the quotes).
  - scope: disc
  - format: characters
- **`DYEAR`**: Year in which the CD was released.
  - scope: disc
  - format: characters (4-digit)
- **`DGENRE`**: Genre (capitalized) of the CD.
  - scope: disc
  - format: characters
- **`TTITLEN`**: Title of the Nth track on the CD (the track number in the keyword should be substituted for the “N”, starting with 0); if there are different artists for the track titles, the track artist and the track title (in that order) should be separated by a ‘/’ with a single space on either side to separate it from the text.
  - scope: tracks
  - format: characters

- **`EXTD`**: Extended CD data (interesting information related to the CD, such as credits etc.).
  - scope: disc
  - format: characters
- **`EXTTN`**: Extended track data (interesting information related to the Nth track, such as the author and other credits, lyrics etc.) for the Nth track (the track number in the keyword should be substituted for the "N", starting with 0).
  - scope: tracks
  - format: characters
- **`PLAYORDER`**: Comma-separated list of track numbers which represent a playlist (generally stripped in non-local databases).
  - scope: disc
  - format: characters

The CDDB file length and the keywords data (except for `DYEAR`) have no size limitation.

Note that there is no specific keyword to indicate the album artist and the album title; the '*artist / title*' scheme is used instead, which applies to both the disc (`DTITLE`) and its tracks (`TTITLEN`). On multi-artist albums the *artist* for the `DTITLE` keyword should be set to *Various*.

Also note that the CDDB format has specific keywords to indicate the release year (`DYEAR`) and the musical genre (`DGENRE`) of the CD, but does not have any specific keyword to indicate the performer(s), composer(s) and songwriter(s). Information about these and other metadata could be entered freely (without any predefined format) using the "extended" fields (`EXTD` and `EXTTN` keywords).

2.3 CDDB: MUSICBRAINZ

MusicBrainz < https://musicbrainz.org/ > ([8]), this also released under the GPL license < https://www.gnu.org/licenses/gpl.html >, is another of the most used CDDB forks which has greatly expanded the original CDDB version (CDDB1) becoming a structured database for music. Since 2005 the project is owned by the MetaBrainz Foundation, a non-profit company. As of 2007, MusicBrainz had a freedb gateway that allows access to their own database. As of May 2019, MusicBrainz contained information about roughly 1.49 million artists, 1.77 million releases, and 20.47 million recordings.

In March 2007 the MetaBrainz Foundation has started the developement of **MusicBrainz Picard** < https://picard.musicbrainz.org/ > ([9]) a cross-platform, free and open-source software application, written in Python and released under the GPLv2 license < https://www.gnu.org/licenses/old-licenses/gpl-2.0.html >, to identify tags (by querying the MusicBrainz database) and organize digital audio recordings.

The ID3v2 ([10]) standard metadata container is supported (all versions, including ID3v2.4), allowing the encoding of strings not only in the ISO-8859-1 (ISO-Latin-1) but also in the UTF-8 character set.

Each CD is identified by a disc ID, calculated with a hash function based on the CD table of contents (but with a different algorithm from that of freedb), and converted into a Base64 encoded ASCII string. The MusicBrainz database ([11]) ([12]) contains information about artists and their recorded works, and the relationships between them. Recorded works entries contain at a minimum the album and track titles, and the length of each track. A lot of additional information can also be stored (such as metadata to indicate the performer(s), composer(s) and songwriter(s), which were missing with CDDB1), also including cover art, acoustic fingerprint and many other metadata.

Basic and advanced MusicBrainz Picard tags (and variables) are a lot and so their right understanding and management is quite complex; for this reason they are not completely listed here (please, refer to the official documentation ([13]) ([14]) for details).

Some of the tags supported by MusicBrainz are particularly significant, as they extend the freedb keywords with the most relevant missing tags, also covering the most commonly used tags.

Among the main basic tags there are:

- **`album`**: Release title.
- **`albumartist`**: Release artist.
- **`albumartistsort`**: Same of `albumartist`, but for sorting.
- **`releasecountry`**: Release country.
- **`date`**: Release date ("YYYY-MM-DD").
- **`originaldate`**: Original release date ("YYYY-MM-DD") intended to provide e.g. the release date of the vinyl version of the CD release.
- **`originalyear`**: Year of the original release date ("YYYY") intended as the release year of the original recording.
- **`catalognumber`**: Release catalog number.
- **`label`**: Official name of the label.
- **`media`**: Release format (CD, SACD, VINYL etc.).
- **`discnumber`**: Disc number (1 to 99) of the disc in this release that the track is on.
- **`totaldiscs`**: Total number (1 to 99) of discs in this release.
- **`title`**: Track title.
- **`artist`**: Track artist.
- **`artistsort`**: Same of `artist`, but for sorting.
- **`isrc`**: Track ISRC code.
- **`tracknumber`**: Track number on the disc.
- **`copyright`**: Copyright holder of the original sound, begin with a year and a space character (e.g.: 2001 (c) Sony).
- **`discid`**: Disc ID of the physical CD release (28 characters: Base64-encoding of the SHA-1 hash calculated from the binary CD TOC data).
- **`comment`**: Short disambiguation comments, used to help distinguish identically named artists, labels and other entities.

while among the main advanced tags there are:

- **`composer`**: Composer for this work, i.e. the artist who wrote the music (not necessarily the lyrics).
- **`composersort`**: Same of `composer`, but for sorting.
- **`conductor`**: The artist who conducted an orchestra, band or choir on this release.
- **`performer`**: An artist that performed on this release.
- **`lyricist`**: The lyricist for this work.
- **`writer`**: The artist responsible for writing the music and/or the words (lyrics, libretto, etc.), when no more specific information is available (composer, lyricist and/or librettist).

Note that MusicBrainz supports musical genres ([15]) only since Picard 2.1 (Dec 21, 2018) as part of its tag system, while earlier versions used folksonomy tags.

Also note how the large number of the available tags actually allows managing a considerable amount of information.

2.4 TOC

The CDDA ([16]) (Compact Disc Digital Audio, also known as Audio CD) standard format for audio compact discs is included in the "Red Book" ([17]) of the so-called "Rainbow Books" series ([18]), firstly released in 1980 by Philips and Sony and adopted as an International Standard in 1987 (IEC 60908); a second edition was published in 1999 replacing the first edition. The official standard is not freely available and licensing must be paid.

The lead-in's subcode of an Audio CD contains repeated copies of the disc's so-called **TOC** (Table Of Contents), which is analogous to the partition table on hard drives as it provides an index of the positions of the tracks. Positions are referenced by absolute timecode, relative to the start of the program area, in MSF format ("`MM:SS:FF`" – minutes, seconds, and timecode frames, being 1 frame = 1/75 of a second).

The TOC of Audio CDs could also contain CD-TEXT information (see the §CD-TEXT chapter) providing metadata about the CD and the tracks it carries on.

Some audio CD rippers – like the free **cdrdao** < http://cdrdao.sourceforge.net/index.html >, released under the GPL license < https://www.gnu.org/licenses/gpl.html > – in conjunction with capable Audio CD readers, might extract CD-TEXT information within a so-called "TOC file", which is usually saved to be available later when burning a "backup" copy of the original CD. The TOC file describes what data is written to the CD-R(W) and allows control over track/index positions, pre-gaps, post-gaps and sub-channel information.

The **TOC file** format ([19]) is consistent with the CD-TEXT specification, but has a different layout structure and also includes additional information through specific statements in the format "`KEYWORD data`".

A TOC file contains an optional header followed by a sequence of track specifications. Comments, starting with `//` reaching until end of line, can be placed anywhere. A so-called CD-TEXT block may be placed in the global disc (header) section, in order to specify general information related both to the whole CD and also for each track description sequence. The disc section must also define a language map (up to 8) used to map a language-number to country codes.

In accordance with the CD-TEXT specification, the character encoding should be ASCII or ISO-8859-1 (ISO-Latin-1).

Any CD-TEXT block, as per the CD-TEXT specification, may contain the following keywords:

- **`TITLE`**: Title of CD or track.
  - scope: disc|tracks
  - format: string
- **`PERFORMER`**: Name of the performer(s).
  - scope: disc|tracks
  - format: string
- **`SONGWRITER`**: Name of the songwriter(s).
  - scope: disc|tracks
  - format: string
- **`COMPOSER`**: Name of the composer(s).
  - scope: disc|tracks
  - format: string
- **`ARRANGER`**: Name of the arranger(s).
  - scope: disc|tracks
  - format: string
- **`MESSAGE`**: Message to the user.
  - scope: disc|tracks
  - format: string
- **`DISC_ID`**: Disc identifier.
  - scope: disc
  - format: string (usually in the format: "XYNNNNN")
- **`GENRE`**: Music genre for the CD.
  - scope: disc
  - format: binary

- **`TOC_INFO1`**: Optional table of contents 1.
    - scope: disc
    - format: mixture of binary data (genre code) and string data
- **`TOC_INFO2`**: Optional table of contents 2.
    - scope: disc
    - format: binary
- **`UPC_EAN`**: Code of the album.
    - scope: disc
    - format: string (13 characters)
- **`ISRC`**: ISRC code of track
    - scope: tracks
    - format: string (12 characters, usually in the format "LLCCCYYNNNNN": 2 country code (alphanumeric), 3 owner code (alphanumeric), 2 year digits (00 to 99), 5 serial number (00000 to 99999))
- **`SIZE_INFO`**: Summary about all CD-TEXT data.
    - scope: disc
    - format: binary

If one of `CD-TEXT`, `TITLE`, `PERFORMER`, `SONGWRITER`, `COMPOSER`, `ARRANGER`, `ISRC` is defined for at least one track or in the disc section, then it must be defined both in the disc section and for each track.

If a `DISC_ID` item is defined in the disc section, then an `ISRC` entry must be defined for each track.

The information on track duration (timecode position in MSF format) – which are not present in the CD-TEXT format specification – must be indicated in each track description sequence, through specific statements such as (see the TOC specification for more):

- **`SILENCE lenght`**: Adds zero audio data of specified length to the current audio track; useful to create silent pre-gaps.
- **`FILE "filename" start [length]`**: Adds the audio data of specified file to the current audio track (it is possible to select a portion of an audio file with `start` ("`MM:SS:FF`") and `length` ("`MM:SS:FF`") which allows non destructive cutting). The first sample of an audio file is addressed with `start` set to 0. If `length` is omitted or set to 0, then all audio data from `start` until the end of file is used.
- **`START start`**: Defines the length ("`MM:SS:FF`") of the pre-gap (position where index switches from 0 to 1). If the start value is omitted the current track length is used. If the current track length is not a multiple of the block length (one audio block corresponds to 2352 bytes, equal to 588 samples) then the pre-gap length will be rounded up to next block boundary. If no start statement is given then the track will not have a pre-gap.
- **`PREGAP pregap`**: Is an alternate way to specify a pre-gap ("`MM:SS:FF`") with zero audio data. It may appear before the first `SILENCE` or `FILE` statement. Either `PREGAP` or `START` can be used within a track specification.
- **`INDEX index`**: Increments the index number at given position ("`MM:SS:FF`") within the track. The first statement will increment from 1 to 2. The position is relative to the real track start, not counting an existing pre-gap.

Note that these keywords, as well as with the CD-TEXT format specification, are consistent with those of the standard CUESHEET format (see the §CUESHEET chapter).

Even the TOC information, as for the CDDB and CD-TEXT data, could be useful when preparing a CUE playlist file (see next chapters) related to the audio CD.

2.5 CD-TEXT

CD-TEXT ([20]) is an extension of the CDDA ([21]) (Compact Disc Digital Audio, also known as Audio CD) standard format for audio compact discs. The CD-TEXT specification ([22]) was included in the MMC standard ([23]) since 1996 (backed by Sony) and added to new revisions of the "Red Book" ([24]) of the so-called "Rainbow Books" series ([25]).

Allowing storage of some audio metadata on CDDA-compliant CDs, CD-TEXT was introduced for the same reasons as the CDDB database; however its format does not include information about album and tracks durations.

When present in audio CDs, the CD-TEXT information is stored in the subchannels R to W (that not all readers are able to read), in the lead-in area (with a maximum data capacity of about 5kB) or in the main program area (with a maximum data capacity of about 31MB) of the disc.

The original Sony authoring tools and specifications supported ASCII and its supersets ISO-8859-1 (ISO-Latin-1) and MS-JIS (japanese Kanji, double byte characters). The ISO-8859-1 (ISO-Latin-1) character encoding is commonly used and supported by CD audio rippers and burners.

The MMC-3 ([26]) specification indicates 13 so-called *types* with *keywords* in the format "`KEYWORD data`" (plus other 3 reserved types without keyword):

- **`TITLE`**: Title of album name or track titles.
    - scope: disc|tracks
    - format: characters
- **`PERFORMER`**: Name of the performer(s).
    - scope: disc|tracks
    - format: characters
- **`SONGWRITER`**: Name of the songwriter(s).
    - scope: disc|tracks

    - format: characters
- **`COMPOSER`**: Name of the composer(s).
    - scope: disc|tracks
    - format: characters
- **`ARRANGER`**: Name of the arranger(s).
    - scope: disc|tracks
    - format: characters
- **`MESSAGE`**: Message from the content provider and/or artist.
    - scope: disc|tracks
    - format: characters
- **`UPC/EAN`**: Code of the album.
    - scope: disc
    - format: characters
- **`ISRC`**: Code of each track.
    - scope: tracks
    - format: characters
- **`DISC_ID`**: Disc identification information.
    - scope: disc
    - format: binary
- **`GENRE`**: Genre identification and information.
    - scope: disc
    - format: binary
- **`TOC_INFO`**: Table of Content information.
    - scope: disc
    - format: binary
- **`TOC_INFO2`**: Second Table of Content information.
    - scope: disc
    - format: binary
- **`SIZE_INFO`**: Size information of the data block.
    - scope: disc
    - format: binary

The character keywords data have no size limitation, since there is an overall limit for the entire CD-TEXT area.

The CD-TEXT information could be used when preparing a CUE playlist file for the CD (see the §CUESHEET chapter).

Note that the CD-TEXT format specification does not define specific keywords neither for the album artist nor for single tracks artist: the `PERFORMER` indicated in the header (disc) section is generally used in place of the album artist, and the `PERFORMER`(s) indicated in the track section for each one of the single tracks are intended as their respective artist. Instead, the keyword `TITLE` is defined to indicate either the disc or tracks titles.

Also note that the CD-TEXT format has specific keywords to indicate both composer(s) (`COMPOSER`) and songwriter(s) (`SONGWRITER`), both used either with respect to album or single tracks.

Other specific keywords indicate relevant information related to the CD, such as the UPC/EAN code of the album (`UPC/EAN`), the ISRC code of each track (`ISRC`), the disc ID (`DISC_ID`) and the musical genre (`GENRE`); while a specific keyword indicating the release year is missing.

2.6 CUESHEET (CDRWIN)

A **cuesheet** (or CUE file, or CUE sheet, or cuesheet file, or .cue etc.) is a text file which uses the **CUESHEET** format ([27]) to describe a CD (or DVD) and the tracks it carries on.

The CUESHEET format was introduced by GoldenHawk Technology for use with **CDRWIN** ([28]) applications, and has since been adopted as the *de facto* standard which is now supported by many optical disc authoring applications and media players. The official CUESHEET syntax and semantics specification is widely accepted to be the Appendix A of the CDRWIN User’s Guide ([29]) ([30]) ([31]) ([32]), which appears around in the year 2000.

The official CUESHEET specification says « *Cuesheet files are standard text (ASCII) files* »; nevertheless the ISO-8859-1 (ISO-Latin-1) character encoding (an ASCII superset) is commonly used and supported, either by CD players and audio applications and by online CDDA databases (freedb, MusicBrainz etc.) also.

The CUESHEET so-called *commands*, in the format “`COMMAND data`” are (see more details in the official specification):

- **`CATALOG mcncode`**: Specify `mcncode` as the UPC/EAN Media Catalog Number (MCN), also known as International Article Number ([33]).
    - scope: disc
    - format: 13 characters
    - restrictions: must appear once in the cuesheet file (should conventionally be the 1st line in the cuesheet, but this is not mandatory)
- **`CDTEXTFILE filename`**: CD-TEXT `filename` (path can be absolute o relative).
    - scope: disc (if present is usually the 2nd cuesheet line)

    - format: characters
- **`FILE filename filetype`**: Media file `filename` (path can be absolute o relative) and `filetype` (`BINARY`|`MOTOROLA`|`AIFF`|`WAVE`|`MP3`).
    - scope: tracks
    - format: characters
    - restrictions: must appear before any other command in the cuesheet file (except for `CATALOG` and `CDTEXTFILE`) – note that in the common use, this rule is conventionally referred as *“before any other command related to the audio tracks contained carried on in that file”*)
- **`FLAGS flags`**: Special track’s subcode `flags` (`DCP`|`4CH`|`PRE`|`SCMS`|`DATA`).
    - scope: tracks
    - format: characters
    - restrictions: could appear after a `TRACK` but before any track’s `INDEX`
- **`INDEX number MM:SS:FF`**: Indexes (or subindexes) `number` (0 to 99) and track position time respect to the current `FILE`, in MSF format (“`MM:SS:FF`” – minutes, seconds, and timecode frames, being 1 frame = 1/75 of a second).
    - scope: tracks
    - format: characters
    - restrictions: see the detailed official CUESHEET specification ([34])
- **`ISRC isrccode`**: Specify `isrccode` as the track’s International Standard Recording Code (ISRC) ([35]).
    - scope: tracks
    - format: 12 characters in “`LLCCCYYNNNNN`” format (2 country code (alphanumeric), 3 owner code (alphanumeric), 2 year (00 to 99), 5 serial number (00000 to 99999))
    - restrictions: could appear after a `TRACK` but before any track’s `INDEX`
- **`PERFORMER performer`**: Name of the performer.
    - scope: disc|tracks
    - format: characters (max 80 characters)
    - restrictions:
      - if `PERFORMER` appears *before any* `TRACK` then `performer` is encoded for the *entire* disc
      - if `PERFORMER` appears *after a* `TRACK` then `performer` is encoded only for the current `TRACK`, not the entire disc
- **`POSTGAP MM:SS:FF`**: Post-gap (gap *after* a track) duration lenght in MSF format (“`MM:SS:FF`” – minutes, seconds, and timecode frames, being 1 frame = 1/75 of a second).
    - scope: tracks
    - format: characters
    - restrictions: could appear only once and after all current track’s `INDEX` commands
- **`PREGAP MM:SS:FF`**: Pre-gap (gap *before* a track) duration lenght in MSF format (“`MM:SS:FF`” – minutes, seconds, and timecode frames, being 1 frame = 1/75 of a second).
    - scope: tracks
    - format: characters
    - restrictions: could appear after a `TRACK`, but before any current track’s `INDEX` commands
- **`REM comment`**: Specify `comment` as a free text comment (often used also to specify unofficial “extended” tags).
    - scope: disc|tracks
    - format: characters (no lenght restriction)
- **`SONGWRITER songwriter`**: Name of the songwriter.
    - scope: disc|tracks
    - format: characters (max 80 characters)
    - restrictions:
      - if `SONGWRITER` appears *before any* `TRACK` then `songwriter` is encoded for the *entire* disc
      - if `SONGWRITER` appears *after a* `TRACK` then `songwriter` is encoded only for the current `TRACK`, not the entire disc
- **`TITLE title`**: Title of the disc or track.
    - scope: disc|tracks
    - format: characters (max 80 characters)
    - restrictions:
      - if `TITLE` appears *before any* `TRACK` commands then `title` (album title) is encoded for the *entire* disc
      - if `TITLE` appears *after a* `TRACK` then `title` (track title) is is encoded only for the CURRENT `TRACK`, not the entire disc
- **`TRACK number datatype`**: Indicates a new track, with track number `number` (1 to 99) and datatype `datatype` (`AUDIO`|…etc…).
    - scope: tracks
    - format: characters (max 80 characters)
    - restrictions: see the detailed official CUESHEET specification ([36])

Note that the `PERFORMER`, `SONGWRITER` and `TITLE` commands are limited to a maximum size of 80 characters, while the `REM` command have no lenght restriction.

Respect to timecode positioning, it is **important** to note what are the implications due to the use restrictions of the `PREGAP`, `INDEX` and `POSTGAP` commands, respect to the `FILE` and `TRACK` commands.

Ripping programs should faithfully extract sound from audio CDs to a single audio file (natively a WAVE file) containing all the audio in the disc, besides separate and standard-compliant TOC and/or CUE files both containing the correct timecode position index of each track, also including the CD-TEXT information if available, and optionally adding metadata by querying online CDDA databases. They might also offer the option of splitting the audio in separate files, and then the gaps between tracks (when present) should be also handled in such a way that to produce a standard-compliant TOC and/or CUE files. The point here is that the official CUESHEET format is designed with the expectation that all of the audio data exists in a single file (CDRWIN only creates cuesheets for that kind of rip), or in separate files (each containing one or more tracks) **but only with the gap portions of the sound placed *before* the beginning of the files (pre-gaps) or eventually removed** (this derives for consistency with the TOC format specification – see the §TOC chapter).

Despite this, some programs – like the proprietary ripper Exact Audio Copy (EAC) – in order to rip audio CDs splitting tracks to single files, or to burn audio CDs starting from sets of splitted audio files, allow relaxing the CDRWIN official restriction on how `FILE` and 'TRACK` commands should be respectively placed between them to correctly handle sound gaps ([37]) ([38]). And here is the problem: this **not-compliant EAC CUESHEET format is *not supported* by many other good pieces of audio software** (rippers, burners, players –- either in Linux and Windows) that instead support the standard (CDRWIN compliant) CUESHEET format. So, if someone shares with you a set of splitted audio tracks among with their not-compliant EAC CUESHEETs, then you are *forced* to use a program supporting this not-compliant EAC CUESHEET format to burn it (or senselessly waste your time to rejoin the audio tracks and accurately recalculate and fix all the time indexes in the cuesheet). Moreover, to play the splitted tracks preserving the right durations and gaps, you are also *forced* to use a player which supports this kind of not-compliant playlist…
This is one of the main reasons why a lot of people (especially those which have and share a large collection of audio CDs and/or wish to listen to high quality audiophile sound with accurate timing and bit-perfect audio data) **prefer a *single* audio file along with a *standard* (CDRWIN) CUESHEET** (also keeping a backup copy of the TOC file, which is the most suitable to be used when burn).

2.7 VORBIS COMMENT (XIPH.ORG)

A **Vorbis comment** ([39]) ([40]) is a metadata container used in the Ogg Vorbis, FLAC, Theora and other audio file formats; it was initially created by the **Xiph.Org** < https://xiph.org/ > Foundation for use with the Ogg Vorbis audio encoding format.

The Vorbis comment format ([41]) defines a minimal set of so-called *comment* fields, intended to quickly identify the audio content of a single file/track rather than be a structured metadata set. There are some proposals for extending the initial format to include other among the most commonly used audio metadata ([42]) ([43]).

A Vorbis comment is a list of *tag* fields in the format "`FieldName=Data`". Any tag name is allowed, and there is no format that the data values must be in. Any comment field may appear any number of times. The number of fields and their length is restricted to about 4,295 millions, but most tag editing applications impose stricter limits.

The field name can be composed of printable ASCII characters (white space included; = and ~ excluded) and is case insensitive; the data are encoded in UTF-8, and so any conforming Unicode string may be used as a value.

These "extensible" comments have found their way into standards besides Ogg Vorbis, such as the Free Lossless Audio Codec (FLAC), so becoming relevant to any digital audio collection.

The minimal list of 15 standard fields in the initial Vorbis specification ([44]) includes:

- **`TITLE`**: Track/work name.
- **`VERSION`**: May be used to differentiate multiple versions of the same track title in a single collection (e.g. remix info).
- **`ALBUM`**: The collection name to which the track belongs.
- **`TRACKNUMBER`**: The track number if part of a specific larger collection or album.
- **`ARTIST`**: The artist generally considered responsible for the work (in popular music this is usually the performing band or singer; for classical music it would be the composer; for an audio book it would be the author of the original text).
- **`PERFORMER`**: The artist(s) who performed the work (in popular music this is typically the same as the `ARTIST` and is omitted; in classical music this would be the conductor, orchestra, soloists; in an audio book it would be the actor who did the reading).
- **`COPYRIGHT`**: Copyright attribution (e.g., '2001 Nobody's Band' or '1999 Jack Moffitt').
- **`LICENSE`**: License information (e.g. 'All Rights Reserved', 'Any Use Permitted', a URL to a license etc.).
- **`ORGANIZATION`**: The name of the organization producing the track (i.e. the 'record label').
- **`DESCRIPTION`**: A short text description of the contents (work description, credits, extra info etc.).
- **`GENRE`**: A short text indication of music genre.
- **`DATE`**: The date the track was recorded or the date the album/collection which the track belongs was recorded.
- **`LOCATION`**: The location where was recorded the track or the album/collection which the track belongs.
- **`CONTACT`**: Contact information for the creators or distributors of the track.
- **`ISRC`**: ISRC number for the track.

Other commonly used *de facto* standard fields ([45]) are:

- **`COMPOSER`**: The composer of the work (in classical music it should be the composer; in popular music etc. this is generally omitted).
- **`TRACKTOTAL`**: The total number of tracks in a collection or album (complements the `TRACKNUMBER` field).
- **`DISCNUMBER`**: The disc number which the track belongs if is part of a specific larger multi-disc album.
- **`DISCTOTAL`**: The total number of discs in a multi-disc album.
- **`SOURCEMEDIA`**: The original media where the track file comes from (CD, DVD, LP…).
- **`PRODUCTNUMBER`**: The UPC/EAN Media Catalog Number (MCN) of the original media where the track file comes from.
- **`CATALOGNUMBER`**: The publisher's catalog number for the original media where the track file comes from.

Note that the standard fields in the Vorbis specification do not contain any timing data; the reason is that they are to be encoded into the same audio files which they refers to, who themselves are tipically intended being as one-per-track.

2.8 MPD TAGS

MPD < https://www.musicpd.org/ > (**Music Player Daemon**) ([46]) is a flexible, powerful, server-side, free and open application for playing music. The first version was released under the GPLv2 license < https://www.gnu.org/licenses/old-licenses/gpl-2.0.html > in 2003 by Max Kellermann.

MPD plays audio files which are listed in its play queue list, organizes playlists and maintains a music database. A client program is needed to interact with the MPD server according to the MPD protocol ([47]); the distribution includes **mpc** < https://www.musicpd.org/clients/mpc/ > (Music Player Client), a simple command line client.

MPD uses a flat file database to maintain the music file information when not running; once the daemon is started, the database is completely load in the system memory avoiding hard disk accesses so as to improve the audio quality of playback. Music files are intended to be stored under the music root directory (usually within sub-directories, one for each album); they are added to the database only when update commands are sent to the server. Playback of arbitrary files is only allowed for local clients which are connected to the server via Unix Domain Sockets.

The client-server model provides several advantages over all-inclusive music players: clients may communicate with the server remotely; the server can be a headless computer located anywhere on the network; different clients can be used for different purposes (a lightweight client for controlling playback, a fully featured client for intensive database searches etc.); several clients and different users can simultaneously use the same database.

The MPD architecture and features make it a great solution for audiophile audio, being able (when running on a fairly powerful and well-configured computer, in conjunction with a good DAC device) to playback high quality audio sound with accurate timings and bit-perfect audio.

The MPD protocol provides that music metadata are stored as *tags* ([48]), in the format "`tag=data`", whose values should be encoded as UTF-8 strings.

The following tags are supported by MPD (version 0.21.6):

- **`artist`**: The artist name (its meaning is not well-defined; see `composer` and `performer` for more specific tags).
- **`artistsort`**: Same as `artist`, but for sorting (this usually omits prefixes such as "The").
- **`album`**: The album name.
- **`albumsort`**: Same as `album`, but for sorting.
- **`albumartist`**: On multi-artist albums, this is the artist name which shall be used for the whole album (the exact meaning of this tag is not well-defined).
- **`albumartistsort`**: Same as `albumartist`, but for sorting.
- **`title`**: The song title.
- **`track`**: The decimal track number within the album.
- **`name`**: A name for the song, which is not the song title (the exact meaning of this tag is not well-defined; it is often used by badly configured internet radio stations with broken tags to squeeze both the artist name and the song title in one tag).
- **`genre`**: The music genre.
- **`date`**: The song's release date, usually a 4-digit year.
- **`originaldate`**: The song's original recording date (available from mpd version >= 0.21).
- **`composer`**: The artist who composed the song.
- **`performer`**: The artist who performed the song.
- **`comment`**: A human-readable comment about the song (the exact meaning of this tag is not well-defined).
- **`disc`**: The decimal disc number in a multi-disc album.
- **`label`**: The name of the label or publisher.

There can be multiple values for some of the MPD tags (for example, MPD may return multiple lines with a `performer` tag).

In addition, other six tags related to the MusicBrainz database are also supported by MPD: **`musicbrainz_artistid`**, **`musicbrainz_albumid`**, **`musicbrainz_albumartistid`**, **`musicbrainz_trackid`**, **`musicbrainz_releasetrackid`**, **`musicbrainz_workid`**.

Note that the **`musicbrainz_albumid`** tag refers to the album id in the MusicBrainz database (where the named `Release Id` tag is mapped with the same *internal* `musicbrainz_albumid` tag). This is particularly relevant for audio CDs and CUE files because, in this case, the internal **`Release Id`** name of MusicBrainz Picard goes to match the **`discid`** tag used by MusicBrainz to identify the physical version of the CD.

MPD does not provide a built-in tag editor; this functionality is handled by clients or external programs, though 3rd party patches adding this functionality to the server.

3 CUESHEETS MORE IN DEPTH

As already mentioned above, the CUESHEET format was introduced by the CDRWIN software, to make available to the users CUE files containing the metadata eventually retrieved by reading the TOC of audio CDs and analyzing the CD-TEXT sections contained therein. Furthermore, by modifying (or creating) CUE files, it was also possible to burn audio CDs writng also the audio metadata indicated in the CD-TEXT section of the TOC.

The close correspondence between the CUESHEET and the TOC/CD-TEXT formats clearly come down from such this implementation, also making it clear that in the CUESHEET format the point of view is the album (the audio CD), which corresponds to a single file containing the digital audio and the related CUE file containing the descriptive metadata of all the tracks on the same CD.

The spread of the CUESHEET format took place thanks to the success of the CDRWIN software, which at the time was widely used as it allowed to quickly and easily back up the PlayStation games. The ability to read and analyze CD TOCs, which in the meantime became a feature common to many other similar applications, made CUE files popular even for burning backup copies of audio CDs. This was accompanied by the diffusion of CDDB services based on large databases made available on the network (freedb and MusicBrainz in particular), which are frequently used by CD authoring and music tagging programs to retrieve the audio CDs metadata and integrate them into related CUE files.

All these circumstances, together with the widespread diffusion of audio CDs, have led CUE files (and therefore the CUESHEET format) to be widely used even today.

The spread of the so-called liquid (digital) music has recently brought a different point of view, no longer based on the album (an audio CD and the corresponding unique audio file) but on individual audio tracks (single audio files, not necessarily corresponding to the tracks of an audio CD) available from many different sources.

Applications for digital music management (among which MPD can be also included) reflect this point of view, in fact having a database whose records consist of sets (also very extensive and detailed, as is for MusicBrainz Picard or the Vorbis comment containers) of attributes (metadata or tags) associated with individual audio files/tracks.

This implies in these programs (as for MPD, which is discussed here) a certain complexity in supporting CUE files in a coherent way with respect to the various standard formats for audio metadata.

A first problem arises from the contraposition “album vs track” due to the opposing approaches, making it necessary to parse CUE files to extract the metadata and match them with the MPD tags avoiding ambiguities and ensuring semantic coherence. This immediately leads to the further problem of effectively defining the correlation between CUE commands and MPD tags.

A further criticality for the correct management of metadata is also present in the case of CUE files associated with classical music CDs (or, more generally, musical genres for which the artist may not correspond to the performer). In this case, the origin of the problem dates back to Sony’s definition of the CD-TEXT format which, presumably for commercial reasons, appears lacking with respect to those musical genres.

More, we must also consider the complexity that occurs in the case of CUE files associated with audio CDs containing tracks of various artists.

At this point in the discussion, it is useful to make some additional references to the structure of a CUE file and to the semantic rules and the common conventions in the use of some CUE commands (refer also to the official CUESHEET syntax and semantics specification ([49]) ([50]) ([51])).

In a CUE file there are usually two main sections:

- ***disc section*** *(header)*, containing information about the entire album;
- ***tracks section***, containing information about the individual tracks.

These sections can be easily identified (see the example below) because the disc section is at the top, while the tracks section follows immediately below, starting the row after the first occurrence of the `FILE` command:

```
REM DISCID "700a0f09"
REM GENRE "Rock"
PERFORMER "Pink Floyd"
TITLE "The Dark Side Of The Moon"
FILE "audiocd.flac" WAVE
  TRACK 01 AUDIO
    PERFORMER "Pink Floyd"
    TITLE "Speak To Me - Breathe"
    PREGAP 00:00:33
    INDEX 01 00:00:00
  TRACK 02 AUDIO
    ...
  ...
FILE ...
  ...
```

A particular trick, that uses the syntax of the **`REM`** command to expand the standard specification, is often used to include certain metadata (often extracted from the TOC/CD-TEXT of audio CDs, or obtained through CDDB database queries) for which there is no specific definition in the standard CUESHEET format. Among the most common use cases are: `REM DISCID`, `REM GENRE`, `REM DATE`. This trick is ***generally and conventionally*** adopted, because in fact it does not break the standard CUESHEET syntax.

In adherence to the standard, as well as for greater compatibility and consistency with different applications, only the genres defined by Sony in the standard CD-TEXT specification ([52]) should be used, which identifies 28 different genres (including `'Not Used'` when no genre applies, and `'Not Defined'` if it is not specified). Unfortunately, a wider non-standard specification (initially introduced by Sony itself with the proprietary version of the CDDB database used by Gracenote) is also often used, which includes over 250 so-called “secondary sub-genres” linked to the original “primary meta-genres”.

Consider now the semantic aspects.

The **`TITLE`** command, when present in the disc section, is referred to as the album title. Instead, when it appears in the tracks section, **`TITLE`** is meant as the title of the track to which it refers.

The **`PERFORMER`** command, when present in the disc section, is *generally and conventionally* used *with reference to the entire album* to indicate the artist (band or singer), or the composer (and not the artist performer) in the case of some genres such as classical music. In the case of various artists, it should take on the value `'Various'` (as for the freedb format specification) or `'Various artists'` (as used in MusicBrainz). In the tracks section the **`PERFORMER`** command should retain the same meaning, but *referring only to the track* for which it is indicated.

Regarding **`PERFORMER`**, the official CUESHEET specification by CDRWIN state these rules: *« If the PERFORMER command appears before any TRACK commands, then the string will be encoded as the performer of the entire disc. If the command appears after a TRACK command, then the string will be encoded as the performer of the current track. »*

The official CUESHEET specification does not offer a specific command to indicate the composer (such a command should have been `COMPOSER`). Therefore, especially in the CUE files associated with classical music CDs, a **`REM COMPOSER`** command is sometimes used, more often *in the tracks section* (this is another standard-compliant use trick of the `REM` command).

The specification provides also a **`SONGWRITER`** command to indicate the text/lirycs writer, or the librettist or the writer of the writings on which the music is inspired. It is rarely present among the metadata that can be obtained from the CDDB databases available online, and is often not covered by CD authoring programs. It is however significant, especially in the case of classical music.

At this point, it is **important** to note that the CUESHEET specification, due to the commands and rules it defines (also regarding the PERFORMER command), does not allow to clearly and unambiguously indicate, *for each single track*, neither the artist to whom the album from which the track comes from is referred, nor the performer and the composer of the track itself. In contrast, MusicBrainz and MPD have specific tags for this: `albumartist` and `artist` in conjunction with `performer` and `composer`.

Moreover, there is *not* any estabilished use of any syntactic CUESHEET form to compensate for these missing commands, as could be `REM ALBUMARTIST` or `REM ARTIST`; indeed, all such forms would result semantically incorrect, being in contrast with the semantic meaning *generally and conventionally* attributed to the `PERFORMER` command.

As a consequence of the above, the CUESHEET specification appears to be ambiguous about the meaning of the `PERFORMER` command, both in absolute sense (it should be intended to indicate the performer in its literal meaning but this does not happen in practice, even for the absence of specific commands to indicate composers and artists, which are instead present among the MPD tags) as well as in relation to the context (cases of classical music albums and multi-artist albums).

For these reasons, especially with CUE files obtained by querying CDDA databases, there are *a lot of different and varied expressions commonly used trying to resolve those ambiguities* (normally without great success) thus be able to insert in the cuesheet the most relevant information about both the album and the tracks at the same time.

For example, in the disc section it is possible to find expressions like:

```
TITLE "Clarinet Concerto KV 622"
PERFORMER ""Mozart - Wesphalian Symphony Orchestra"
```

```
TITLE "Symphony No. 5"
PERFORMER "Mahler - London Philarmonic Orchestra, C. Abbado (director)"
```

```
TITLE "Chopin Nocturnes"
PERFORMER "Frédéric Chopin / Nocturnes - Maria João Pires (piano)"
```

or, in the tracks section, like:

```
TITLE "Requiem: ii. Offertoire"
PERFORMER "Gabriel Fauré - Academy of St Martin in the Fields, Sir Neville Marriner (director)"
```

```
TITLE "Nocturne for piano No. 13"
PERFORMER "Frédéric Chopin - Maria João Pires (piano)
```

```
TITLE "Money"
PERFORMER "Pink Floyd (Waters)
```

Finally, it is worth highlighting the importantance that CUE files were correctly set up, *always respecting the syntax of the standard CUESHEET format* (even when using tricky commands), and also *paying attention to the semantic meaning of the CUE commands as well as to the generally accepted conventions of use*. In relation to this aspect, see the examples below (note the use of the `TITLE`, `PERFORMER` and `REM COMPOSER` commands):

- non-classical music:

```
REM DISCID "700a0f09"
REM GENRE "Rock"
TITLE "The Dark Side Of The Moon"
PERFORMER "Pink Floyd"
FILE "audiocd.flac" WAVE
  TRACK 01 AUDIO
    TITLE "Speak To Me - Breathe"
    REM COMPOSER "Mason - Waters, Gilmour, Wright"
    PERFORMER "Pink Floyd"
    PREGAP 00:00:33
    INDEX 01 00:00:00
  TRACK 02 AUDIO
    TITLE "On The Run"
    REM COMPOSER "Gilmour, Waters"
    PERFORMER "Pink Floyd"
    INDEX ...
  ...
```

- classical music:

```
REM DISCID "c610500e"
REM GENRE "Classical"
TITLE "Fauré Requiem"
PERFORMER "Gabriel Fauré"
FILE "audiocd.flac" WAVE
  TRACK 01 AUDIO
    TITLE "Requiem: i. Introit et Kyrie"
    REM COMPOSER "Gabriel Fauré"
    PERFORMER "Academy of St Martin in the Fields, Sir Neville Marriner (director)"
    INDEX 01 00:00:00
  TRACK 02 AUDIO
    TITLE "Requiem: ii. Offertoire"
    REM COMPOSER "Gabriel Fauré
    PERFORMER "Academy of St Martin in the Fields, Sir Neville Marriner (director);"
    INDEX ...
  ...
```

- multi-artist:

```
REM DISCID=e8090810
REM GENRE "Pop Music"
TITLE="Pulp Fiction - Music From The Motion Picture"
PERFORMER "Various"
FILE "audiocd.flac" WAVE
  TRACK 01 AUDIO
    TITLE "Misirlou"
    PERFORMER "Dick Dale And His Del-Tones"
    INDEX 01 00:00:00
  TRACK 02 AUDIO
    TITLE "Royale With Cheese (Dialogue)"
    PERFORMER "John Travolta, Samuel L. Jackson"
    INDEX ...
  ...
```

4 CORRELATIONS BETWEEN CUESHEET AND MPD METADATA

In the following, the correlations between the metadata related to the descriptive attributes of the audio tracks defined by CUE commands and MPD tags are treated. The metadata related to audio files and track durations will not be examined, if not incidentally, and not even those related to the track numbers and to the sorting and searching keys within the database.

4.1 METADATA FOR ALBUM, ALBUMARTIST, TITLE, ARTIST, PERFORMER AND COMPOSER

The correlations that involve the greatest semantic problems discussed in the previous chapter are examined here below, also taking into consideration the three examples given at the end of the same chapter.

In correspondence with the `TITLE`, `PERFORMER` and `REM COMPOSER` CUE commands, MPD supports the `album`, `albumartist`, `title`, `artist`, `performer` and `composer` tags. Thanks to the latter and despite the further difficulty due to the absence of a 1:1 correspondence, it is still possible, by appropriately working with the correlations between the tags, to solve many of the ambiguities mentioned above.

Before proceeding with the discussion, it is therefore opportune to recall again the MPD specification (already previously examined in the §MPD chapter) regarding these tags:

- `album`: The album name.
- `albumartist`: On multi-artist albums, this is the artist name which shall be used for the whole album (the exact meaning of this tag is not well-defined).
- `title`: The song title.
- `performer`: The artist who performed the song.
- `composer`: The artist who composed the song.
- `artist`: The artist name (its meaning is not well-defined; see `composer` and `performer` for more specific tags).

Note how **`albumartist` and `artist`** ***are both « not well-defined »*** in the MPD specification, and how **`artist`** ***should be referred to `composer` and `performer` as « more specific tags »***.

It is also useful to recall the Vorbis comment clarification about the semantic meaning of the `ARTIST` tag, which is intended as: « ***The artist generally considered responsible for the work (in popular music this is usually the performing band or singer; for classical music it would be the composer)*** ».

Now, as a result of what was discussed in the previous chapter, it is clear that in the **disc section** of the cuesheet:

- **`TITLE`**, with the meaning of: ***title of the album***, must be associated with the **`album`** tag *for all the tracks* on the album;
- **`PERFORMER`**, with the meaning of ***artist to whom the entire album is referred***, should be normally associated with the **`albumartist`** tag *for all tracks* on the album;
- **`REM COMPOSER`** *if present AND in the ONLY case of the* `Classical` *genre* (or more generally, the *musical genres for which the artist may not correspond to the performer*), should be associated to the **`albumartist`** tag *for all the tracks* on the album (rather than `PERFORMER`), with the same meaning of ***artist to whom the entire album is referred***.

Furthermore, in the **tracks section** of the cuesheet:

- **`TITLE`**, with the meaning of ***title of the track***, must be associated with the **`title`** tag for the *specific track* it refers to;
- **`PERFORMER`**, with its literal meaning of ***performer***, must be associated with the **`performer`** tag for the *specific track* it refers to, *except* when `PERFORMER` is set to `Various` (or `Various Artists`, although many other variants or acronyms can be found, depending on the language and the common use);
- **`REM COMPOSER`**, with its literal meaning of ***composer***, should be associated with the **`composer`** tag for the *specific track* it refers to;
- **`PERFORMER`** – or **`REM COMPOSER`** *in the ONLY case of the* `Classical` *genre* (or more generally, the *musical genres for which the artist may not correspond to the performer*) – should be normally associated with the **`artist`** tag for the *specific track* they refers to.

In light of all these recalls and considerations, it is now finally possible to satisfactorily define the **correlation rules** to be implemented when parsing (in cascade) `PERFORMER` and `REM COMPOSER` commands in CUE files, in order to set the `albumartist`, `artist`, `performer` and `composer` MPS tags in an optimal way:

– DISC SECTION

- **if `PERFORMER` *IS* specified in the disc section, then it MUST be associated to `albumartist` for each track**;
- **if `REM COMPOSER` *IS* specified in the disc section *AND only* for the musical genres for which the artist may not correspond to the performer** (as is for `Classical`), **then it should be associated to `albumartist` for each track (instead of `PERFORMER`)**;

– TRACKS SECTION

- **if `PERFORMER` *IS* specified at a specific track, then it must be associated to both `artist` AND `performer` for that track**;
- **if `PERFORMER` *IS NOT* specified at a specific track**:
  - **`artist` for that track must be done coincide with `albumartist`** (which is intended for the whole album) – ***except* if it *is* a *various artists* album: in this case `artist` should be set to `Unknown`** (which is commonly used in CDDB databases such as freedb and MusicBrainz when the artist of a certain track is not specified);
  - **`performer` for that track should be set to `Unknown`**;
- **if `REM COMPOSER` *IS* specified at a specific track, then it must be associated to `composer` for that track**;
- **if `REM COMPOSER` *IS NOT* specified at a specific track then**:
  - **if a `REM COMPOSER` *is* present in the disc section** (which is intended for the whole album) **it should associated to `composer` for that track**;
  - ***elsewhere* `composer` for that track should be set to `Unknown`**.

When all the correlation rules between CUE commands and MPD tags are well established during the parsing of syntactically corrected and semantically well-defined CUE files, then **the MPD tags** (`album`, `albumartist`, `title`, `artist`, `performer` and `composer`) **will be set with semantically appropriate values** (at least as far as they do are semantically appropriated in the input CUE file), **even for musical genres where the artist does not normally correspond to the performer** (as for `Classical`), **and for multi-artist albums** (`Various` artists) also.

Moreover, to try to increase the chances that clients give semantically congruous results when parsing, it would be **always preferable to specify `PERFORMER` in *both* the disc section of the CUESHEET and for *all* the tracks in the tracks section**. Similarly, when it should be indicated for a single track, it would be **always preferable to specify `COMPOSER` for *all* the tracks** in the track section.

In a later chapter a §**A PSEUDOCODE PROPOSAL** is reported, which implements the parsing of CUE files with the correlation rules as indicated here above.

In addition, it should be noted that, in order to be able to perform parsing at best, it might also be convenient to allow users to specify (typically through the configuration file) a ***list of musical genres for which the artist should correspond to the composer rather than to the performer***, as is for `Classical` (which would be the default value). Similarly, given the many acronyms and linguistic conventions which are used in different countries, it might also be convenient to manage a ***list of synonyms for* `Various`** when the case of multi-artist albums.

Finally, also consider how mpd and clients generally behave in this regard.

Actually, MPD (version 0.21.6) and the mpc client (version 0.30) perform the parsing of CUE files according to a different procedure than that shown above. In fact, both from the code analysis (source file: `CueParser.cxx`) and from specific tests carried out with the mpc client (command: `mpc -f`), the way they actually behaves when parsing CUE files turns out to be the following:

- cuesheet's `PERFORMER` values are never assigned to the MPD's `performer` tag;
- if the `PERFORMER` command appears in the header (album) section at the top of the cuesheet, then its value is assigned to the `albumartist` tag, which refers to the whole album;
- if a `PERFORMER` command appears in the track section of the cuesheet, then its value is assigned to the `artist` tag for the corresponding track;
- if the `REM COMPOSER` command appears in the header (album) section at the top of the cuesheet, then its value is assigned to the `composer` tag for *ALL* the tracks in the album, superseeding any `REM COMPOSER` command in the track section of the cuesheet;

- if a `REM COMPOSER` command appears in the track section of the cuesheet *AND* there is *NOT* any `REM COMPOSER` command in the header (album) section, then its value is assigned to the `composer` tag for the corresponding track.

Based on the conclusions reached earlier, this parsing procedure does not fully solve the discussed ambiguities, and does not specifically address the case of genres such as `Classical` and that of multi-artist albums; so it do not appear to be suitable for optimally managing tags in CUE files.

It should also be noted that many of the currently available MPD clients parse CUE files using their own procedures, thus not relying on MPD in the management of CUE files. From several tests carried out, at the moment it does not seem that there are MPD clients that support CUE files in a really proper way.

Among the graphic clients Cantata < https://github.com/CDrummond/cantata > seems to be generally indicated as the one with the best support for the CUE files, although not yet satisfactory.

4.2 CORRELATIONS FOR OTHER STANDARD METADATA

In addition to the metadata examined in the previous section, it is also necessary to establish how to parse CUE files to manage both the additional CUE commands which are defined in the official CUESHEET specification (`REM`, `CATALOG`, `ISRC` and `SONGWRITER`) and the other tags in the MPD specification (`comment`, `genre`, `date`, `originaldate`, `disc` and `label`).

– Correlations between standard MPD tags and standard CUE commands

Among all these tags, there is a direct correspondence only between **`comment`** and **`REM`** (comment), while MPD does *not* have a corresponding tag either for `CATALOG` (to specify the unique CD's UPC/EAN Media Catalog Number – MCN) or for `ISRC` (the unique track's International Standard Recording Code – ISRC).

The ideal solution would be to **extend the MPD specification**, introducing new specific tags for this purpose, which could simply be **`catalog`** and **`isrc`**, making them correspond to **`CATALOG`** and **`ISRC`** respectively. Another acceptable solution, though less desirable, could also be to enter these metadata by appending them to the actual value of the `comment` tag (e.g. by using a format such as `CATALOG:catalog. ISRC:isrc.`).

Not even for `SONGWRITER` (the text/lirycs writer, or the librettist, or the writer of the writings on which the music is inspired) there is a corresponding MPD tag.

Again, also in this case, the ideal solution would be to introduce a new MPD tag, which could simply be **`songwriter`**, putting it in direct correspondence with **`SONGWRITER`**. Otherwise, as an alternative, MPD could append `SONGWRITER` to the actual value of `comment` or simply ignore it (as is currently done).

In summary, the new *"extended"* MPD tags which have just been proposed here to be directly matched to the respective CUE commands, are: **catalog**, **isrc**, **songwriter**.

– Standard MPD tags for which there are no corresponding standard CUE commands

The opposite situation occurs for those MPD tags (**`genre`**, **`date`**, **`originaldate`**, **`disc`** and **`label`**) for which there are no corresponding CUE command in the official CUESHEET specification.

However, in the disc section of CUE files, it is always ***conventionally*** present a **`REM GENRE`** command (the music genre), and very often there is also a **`REM DATE`** command (the release CD/album date); both them can be considered as *de facto* included in the standard CUESHEET format. It is therefore correct to manage these CUE commands simply correlating them to the corresponding **`genre`** and **`date`** MPD tags respectively:

- **`REM GENRE genre`**: Music genre.
  - scope: disc
  - format: characters (see: Sony CD-TEXT genres list in the CD-TEXT format specification ([53]); when no genre applies, should be set to: 'Not Used'; if missing, should be set to: 'Not Defined')
- **`REM DATE yyyy-mm-dd`**: Album/collection release date.
  - scope: disc
  - format: characters (4-10 digits, in "YYYY[-MM-DD]" format – examples: "1976-05-21", "1976")

More rarely, the disc section of CUE files contains specific commands corresponding to the `originaldate` (the original recording date), `disc` (the disc number in a multi-disc album) and `label` (the name of the label or publisher) MPD tags; furthermore, it is possible to find several different CUE syntax forms for these tags, using the tricky construct **`REM`** ***`tag`***.

A specific deepening should be done for the MPD tag `originaldate`, whose precise meaning and format is not entirely clear. The MPD documentation defines this tag as the "original recording date", but it is not clear how it should be evaluated: as a year or as a real date. Indeed, MusicBrainz distinguishes in fact between `originaldate` (which is a real date, in the format "`YYYY-MM-DD`") to `originalyear` (which is a year, in the format "`YYYY`"). Since the `REM DATE` CUE command is generally used in the "`YYYY`" format to indicate the year of release (deriving from the keyword `DYEAR` used in the CDDB databases), it certainly appears correct to maintain the same approach introducing a `REM ORIGINALDATE` CUE command with respect to the original recording date (or year), eventually adding also an additional `REM ORIGINALYEAR` command with respect to the original recording year.

Based on what was discussed above, the following are proposed here to handle the `originaldate`, `disc` and `label` MPD tags:

- **`REM ORIGINALDATE yyyy-mm-dd`** (MPD, MusicBrainz): Album/collection original recording date.
  - scope: disc
  - format: characters (4-10 digits, in "YYYY[-MM-DD]" format – examples: "1976-05-21", "1976")

- **`REM ORIGINALYEAR yyyy`** (MusicBrainz): Album/collection original recording year.
    - scope: disc
    - format: characters (4 digits, in “YYYY” format – example: “1976”)
- **`REM DISC nn`** (MPD) or **`REM DISCNUMBER nn`** (MusicBrainz): Disc number in a multi-disc album release.
    - scope: disc
    - format: characters (1-2 digits, in “[N]N” format, 1 to 99 – example: “2” or “02”)
- **`REM LABEL label`** (MPD, MusicBrainz): Record label.
    - scope: disc
    - format: characters (alphanumeric, of max 80 characters – example: “Sony Classical Records”)

In addition to these, despite the fact that there is a direct correlation between `comment` and `REM` (comment), it seems to be useful to propose – either to facilitate the CUE files parsing by some client applications and for better compatibility with MPD – the introduction of an additional tricky CUE command to handle the correspondence with the `comment` MPD tag:

- **`REM COMMENT comment`**: Free text comment.
    - scope: disc|tracks
    - format: characters (alphanumeric – example: “Digitally remastered 24bit”)

It should be remarked how **however it is always necessary to handle very carefully** of all these CUE commands when parsing CUE files, in order to detect all the most widespread uses, and then match them correctly to the corresponding MPD tags.

Please, see also the detailed §**REFERENCE SUMMARY FOR CUESHEET AND MPD AUDIO METADATA** chapter which follows later.

4.3 ADDITIONAL NON-STANDARD EXTENDED METADATA

In addition to all the metadata that have already been discussed, CUE files there can contain many other information, specified either as standard comments (through the tricky `REM` use) or even through “special” *non-standard extended commands*, having disparate formats and not included in the official CUESHEET specification.

Among all these, it is necessary to identify the most significant ones that are more consistent with the standard formats for audio metadata; that is to say those for which there is some correspondence in the most commonly used format specifications in data sources generally consulted in preparing CUE files (in particular CD-TEXT, freedb, MusicBrainz and Vorbis comment).

The following are identified and proposed here:

- **`REM DISCID discid`** (freedb): Freedb discid.
    - scope: disc
    - format: characters (8 hex 32bit digits, in “CCSSSSNN” format: CC = tracks based checksum mod 255, SSSS = total time of the CD in seconds, NN = number of CD tracks)
- **`REM ALBUMID discid`** (MusicBrainz): MusicBrainz discid of the physical CD release.
    - scope: disc
    - format: characters (28 characters: Base64-encoding of the SHA-1 hash calculated from the binary CD TOC data)
- **`REM MEDIA mediatype`** (MusicBrainz) or **`REM SOURCEMEDIA mediatype`** (Vorbis comment): Media where the original album/collection comes from.
    - scope: disc
    - format: characters (should be one of: Compact Disc (or CD), Digital Video Disc (or DVD), Long Playing (or LP), Music Cassette (or MC) etc.)
- **`REM TOTALDISCS number`** (MusicBrainz) or **`REM DISCTOTAL number`** (Vorbis comment): Total number of discs in a multi-disc album release.
    - scope: disc
    - format: characters (1-2 digits, from 1 to 99, in “[N]N” format)
- **`REM COPYRIGHT year_and_holder`** (MusicBrainz, Vorbis comment): Year and copyright holder of the original disc or media or recording.
    - scope: disc
    - format: characters (alphanumeric, should be of max 80 characters, must begin with a year and a space character – example: “2001 (c) Sony”)
- **`REM LOCATION location`** (Vorbis comment): Location where recorded.
    - scope: disc
    - format: characters (alphanumeric, should be of max 80 characters – example: “Abbey Road Studios, London”)

Also in the case of these metadata, the ideal solution would be to **extend the MPD specification** by introducing new specific tags to which directly correlate the respective CUE commands, obviously taking care to parse CUE files in order to properly identify and manage those tags. Again, it would also be acceptable, but less desirable, the solution to append such metadata at the end of the `comment` tag, clearly indicating their meaning and separating them appropriately from one another.

The new *“extended”* MPD tags proposed here to directly match the corresponding *“extended”* CUE commands (retaining their semantic meaning) could simply be, respectively: **`discid`**, **`albumid`**, **`media`** (or **`sourcemedia`**), **`totaldiscs`** (or **`disctotal`**), **`copyright`**, **`location`**.

Note how the new `albumid` tag is proposed here although the `REM ALBUMID` might have a correspondence with the `musicbrainz_albumid` MPD tag: this is in order to keep the `musicbrainz_albumid` tag available for other possible purposes (for example, for any queries to the MusicBrainz database).

Please, see also the detailed following §**REFERENCE SUMMARY FOR CUESHEET AND MPD AUDIO METADATA** chapter.

5 REFERENCE SUMMARY FOR CUESHEET AND MPD AUDIO METADATA (WITH PROPOSALS)

Based on everything discussed so far, it is now possible to derive the reference summary for CUE and MPD audio metadata (both standard and extended) which is presented below; proposals to standardize and extend both the CUE commands and the MPD tags as per the discussion above, there are also reported.

REFERENCE SUMMARY FOR CUESHEET AND MPD AUDIO METADATA						
CUESHEET REFERENCE SAMPLE	**CUESHEET REFERENCE (*)**		**MPD REFERENCE (*)**		**MEANING**	**NOTES**
REM DISCID “850aa60a”	***REM DISCID*** *“discid_freedb”*	*(proposed ext)*	***discid***	*(proposed ext)*	Freedb (cddb1) DISCID.	8 hex 32bit digits, in “CCSSSSNN” format: CC = tracks based mod 255 checksum, SSSS = total time of the CD in seconds, NN = number of CD tracks.
REM ALBUMID “XzPS7vW.HPHsYemQh0HBUGr8vuU-”	***REM ALBUMID*** *“discid_musicbrainz”*	*(proposed ext)*	***albumid***	*(proposed ext)*	MusicBrainz DISCID.	28 characters (Base64-encoding of the SHA-1 hash calculated from the binary CD TOC data).
REM GENRE “Folk”	**REM GENRE** “genre”	(de facto std)	**genre**		Music genre.	See: Sony CD-TEXT genres list in the CD-TEXT format specification. When no genre applies, should be set to: ‘Not Used’; if missing, should be set to: ‘Not Defined’.
REM DATE “1995”	**REM DATE** “YYYY[-MM-DD]”	(de facto std)	**date**		Album/collection release date.	4-10 digits, in “YYYY[-MM-DD]” format.
REM ORIGINALDATE “1976-05-21”	***REM ORIGINALDATE*** *“YYYY[-MM-DD]”*	*(proposed ext)*	**originaldate**		Album/collection original recording date.	4-10 digits, in “YYYY[-MM-DD]” format.
REM ORIGINALYEAR “1976”	***REM ORIGINALYEAR*** *“YYYY”*	*(proposed ext – optional)*	**originaldate**		Album/collection original recording year.	4 digits, in “YYYY” format.
REM MEDIA “CD” *or:* REM SOURCEMEDIA “CD”	***REM MEDIA*** *“mediatype”* or: *REM SOURCEMEDIA “mediatype”*	*(proposed ext)*	***media***	*(proposed ext)*	Media where the original album/collection comes from.	Should be one of: Compact Disc (or CD), Digital Video Disc (or DVD), Long Playing (or LP), Music Cassette (or MC) etc.
REM DISC “1” *or:* REM DISCNUMBER “1”	***REM DISC*** *“[N]N”* or: *REM DISCNUMBER “[N]N”*	*(proposed ext)*	**disc**		Disc number (relevant in a multi-disc album release).	1-2 digits, from 1 to 99, in “[N]N” format.
REM TOTALDISCS “2” *or:* REM DISCTOTAL “2”	***REM TOTALDISCS*** *“[N]N”* or: *REM DISCTOTAL “[N]N”*	*(proposed ext)*	***totaldiscs***	*(proposed ext)*	Total number of discs (relevant in a multi-disc album release).	1-2 digits, from 1 to 99, in “[N]N” format.
CATALOG “5017615845822”	**CATALOG** “NNNNNNNNNNNNN”		***catalog***	*(proposed ext)*	UPC/EAN Media Catalog Number (MCN).	13-digit code in EAN-13 format, containing a prefix ‘0’ followed by a 12-digit UPC-A code identifier. Should conventionally be the 1st line in the cuesheet, but this is not mandatory.
REM LABEL “Sony Classical Records”	***REM LABEL*** *“label”*	*(proposed ext)*	**label**		Record label.	Alphanumeric (should be of max 80 characters in the cuesheet).
REM COPYRIGHT “2001 (c) Sony”	***REM COPYRIGHT*** *“YYYY holder”*	*(proposed ext)*	***copyright***	*(proposed ext)*	Year and copyright holder of the original sound.	Alphanumeric (should be of max 80 characters in the cuesheet). Must begin with a year and a space character.
REM LOCATION “Abbey Road Studios, London”	***REM LOCATION*** “location”	*(proposed ext)*	***location***	*(proposed ext)*	Location where recorded.	Alphanumeric (should be of max 80 characters in the cuesheet).
REM “Recorded live”	**REM** “comment”		**comment**		Any free text comment (work description, credits, editor, ripper sw, extra info etc.).	Alphanumeric (no lenght restriction in the cuesheet - recommended: max 80 characters).
REM COMMENT “Recorded live”	***REM COMMENT*** *“comment”*	*(proposed ext – optional)*	**comment**		Any free text comment (work description, credits, editor, ripper sw, extra info etc.).	Alphanumeric (no lenght restriction in the cuesheet - recommended: max 80 characters).
TITLE “Verdi Requiem”	**TITLE** “title”		**album**		Album/collection title.	Alphanumeric (should be of max 80 characters in the cuesheet).
REM COMPOSER “Giuseppe Verdi”	***REM COMPOSER*** *“composer”*	*(proposed ext)*	**composer | albumartist** (**)		Album music/songs composer (Classical) | album artist/writer (others).	Alphanumeric (should be of max 80 characters in the cuesheet).

REFERENCE SUMMARY FOR CUESHEET AND MPD AUDIO METADATA				
CUESHEET REFERENCE SAMPLE	**CUESHEET REFERENCE (*)**	**MPD REFERENCE (*)**	**MEANING**	**NOTES**
PERFORMER “BBC Symphony Orchestra & Chorus, A. Toscanini (director)”	**PERFORMER** “performer”	**performer | albumartist** (**)	Album/collection soloist(s), ensamble/orchestra, conductor (Classical) | album artist band/singer (others).	Alphanumeric (should be of max 80 characters in the cuesheet). On multi-artist albums should be set to: “Various”.
SONGWRITER “Catholic Mass for Dead and burial (based on)”	**SONGWRITER** “songwriter”	***songwriter*** *(proposed ext)*	Album/collection librettist or author of texts on which is based/inspired (Classical) | album songs lirycist (others).	Alphanumeric (should be of max 80 characters in the cuesheet).
	FILE …		*First line in the cuesheet’s tracks section…*	
	TRACK 01 AUDIO		…	
TITLE “Requiem: i. Introit et Kyrie”	**TITLE** “title”	**title**	Track title.	Alphanumeric (should be of max 80 characters in the cuesheet).
REM COMPOSER “Gabriel Fauré”	***REM COMPOSER*** *“composer”* *(proposed ext)*	**composer | artist** (**)	Track music/song composer (Classical) | track artist/writer (others).	Alphanumeric (should be of max 80 characters in the cuesheet).
PERFORMER “Joyce DiDonato (mezzo-soprano)”	**PERFORMER** “performer”	**performer | artist** (**)	Track soloist(s), ensamble/orchestra, conductor (Classical) | track artist band/singer (others).	Alphanumeric (should be of max 80 characters in the cuesheet).
SONGWRITER “Lorenzo Da Ponte (librettist)”	**SONGWRITER** “songwriter”	***songwriter*** *(proposed ext)*	Track librettist or author of texts on which is based/inspired (Classical) | track lirycist (others).	Alphanumeric (should be of max 80 characters in the cuesheet).
ISRC “DEGP31200838”	**ISRC** “LLCCCYYNNNNN”	***isrc*** *(proposed ext)*	ISCR code number for the track.	12 alphanumeric characters, in “LLCCCYYNNNNN” format: 2 country code [0-9A-Z], 3 owner code [0-9A-Z], 2 year (00 to 99), 5 serial number (00000 to 99999).
REM “Remastered 24bit”	**REM** “comment”	**comment**	Any comment (track description, credits, editor, ripper sw, extra info etc.).	Alphanumeric (no lenght restriction in the cuesheet - recommended: max 80 characters).
REM COMMENT “Remastered 24bit”	***REM COMMENT*** *“comment”* *(proposed ext – optional)*	**comment**	Any comment (track description, credits, editor, ripper sw, extra info etc.).	Alphanumeric (no lenght restriction in the cuesheet - recommended: max 80 characters).
	PREGAP …			
	INDEX …			
	POSTGAP …			
	… next tracks follow below …			

NOTES:
(*) All the CUE command and MPD tags referenced here are in **bold**; proposed extensions are also ***italics***.
(**) For the **correlation rules** between ‘PERFORMER’ and ‘REM COMPOSER’ with ‘albumartist’, ‘artist’, ‘performer’ and ‘composer’, see chapter §METADATA FOR ALBUM, ALBUMARTIST, TITLE, ARTIST, PERFORMER AND COMPOSER.

5.1 CUESHEET EXAMPLE

The example below is drawn up in accordance with the CUESHEET reference above, and can be used as a sort of *template structure* when editing a CUESHEET:

```
CATALOG "5099964052922"
REM DISCID "930d950a"
REM ALBUMID "rq2kmk2sXobXaiKNJmGB7h0nuPo-"
REM GENRE "Classical"
REM DATE 2010
REM ORIGINALDATE 2010
REM MEDIA "CD"
REM DISC 1
REM TOTALDISCS 1
REM LABEL "Warner Classics"
REM COPYRIGHT "2010 EMI Records Ltd."
REM COMMENT "Recorded November 8, 2010; Released December 7, 2010"
REM LOCATION "Sala Santa Cecilia, Auditorium Parco della Musica, Roma"
PERFORMER "Gioacchino Rossini"
SONGWRITER "Traditional liturgical sequence text often ascribed to Jacopone da Todi (ca. 1230–1306)"
TITLE "Rossini: Stabat Mater"
FILE "gioacchino_rossini-stabat_mater-pappano-2010.flac" WAVE
  TRACK 01 AUDIO
    TITLE "Stabat Mater: I. Introduzione: Stabat Mater dolorosa"
    REM COMPOSER "Gioacchino Rossini"
    PERFORMER "Anna Netrebko (soprano), Joyce DiDonato (mezzo-soprano), Lawrence Brownlee (tenor), Ildebrando d'Arc
angelo (bass), Orchestra e Coro dell’Accademia Nazionale di Santa Cecilia, Antonio Pappano (director)"
    INDEX 01 00:00:00
  TRACK 02 AUDIO
    TITLE "Stabat Mater: II. Aria: Cujus animam gementem"
    REM COMPOSER "Gioacchino Rossini"
    PERFORMER "Lawrence Brownlee (tenor), Orchestra dell’Accademia Nazionale di Santa Cecilia, Antonio Pappano (dir
ector)"
    INDEX 01 09:29:23
  TRACK 03 AUDIO
    TITLE "Stabat Mater: III. Duetto: Quis est homo"
    REM COMPOSER "Gioacchino Rossini"
    PERFORMER "Anna Netrebko (soprano), Joyce DiDonato (mezzo-soprano), Orchestra dell’Accademia Nazionale di Santa
Cecilia, Antonio Pappano (director)"
    INDEX 01 6:16:47
  TRACK ...
    ...
    ...
```

It is very important to recall that the **structure for each of the tracks** has the general schema:

```
  ...
  TRACK NN AUDIO
    TITLE "Track title"
    REM COMPOSER "Composer"
    PERFORMER "Performer"
    SONGWRITER "Track lyrics or texts author or librettist"
    REM COMMENT "Free text comment"
    ISRC “LLCCCYYNNNNN”
    PREGAP MM:SS:FF
    INDEX NN MM:SS:FF
    INDEX ...
    INDEX ...
    POSTGAP MM:SS:FF
  ...
```

It is important to remember that (please refer to the correlation rules stated in the §CORRELATIONS BETWEEN CUESHEET AND MPD METADATA chapter):

- **in the disc section** of a CUESHEET, **`PERFORMER`** should indicate ***the artist (performer or composer) to whom the entire album is referred*** (which is normally associated with the **`albumartist`** MPD tag); on **multi-artist albums** it should be set to “**Various**” (or “Various artists”);
- when present **in correspondence of a specific track** of the CUESHEET, **`PERFORMER`** (with its literal meaning) should indicate ***the artist performing that specific track*** (and is normally associated with the **`artist`** MPD tag).

Moreover, also remember that:

- in the same CUESHEET, **more than one `FILE` may be present**: in this case, each `FILE` ***must*** precede all its contained `TRACK`s.

Finally, note that clients implement CUESHEET support in different ways and with different algorithms. Thus, to try to increase the chances that clients give semantically congruous results when parsing, it would be **always preferable to specify `PERFORMER` in *both* the disc section of the CUESHEET and for *all* the tracks in the tracks section**. Similarly, when it should be indicated for a single track, it would be **always preferable to specify `COMPOSER` for *all* the tracks** in the track section.

For details about the syntax, please refer also to the §CUESHEET chapter.

6 PSEUDOCODE PROPOSAL

The following is a pseudocode proposal, developed on the whole analysis and discussion carried out in the previous chapters, which implements the parsing of CUE files commands in relation to the MPD tags. This pseudocode is released under the GNU General Public License < https://www.gnu.org/licenses/gpl.html > , Version 3 or any later version.

Note that this pseudocode assumes that users can specify, through specific options in the configuration files (of MPD or of the MPD client), both a list of genres for which the the artist does not normally correspond to the performer (as is for `Classical`) and a list of synonyms or equivalent alternatives for ‘Variuos’ (as is needed for multi-artist albums). However, a valid alternative pseudocode is also indicated in case these options are not supported.

In the next section in this chapter, some synthetic examples will also be provided, to show what result should be obtained with this implementation, in relation to some different typical situations.

6.1 PSEUDOCODE

```
//
// cuesheet-parsing
//
// Pseudocode -- CUE file metadata parser for MPD
//
// Version: 2.0  2019-05-08
//
// Copyright (c) 2018+ alexus. Released under the GNU General Public License, Version 3.
//
// This program is free software: you can redistribute it and/or modify it under the
// terms of the GNU General Public License as published by the Free Software Foundation,
// either version 3 of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
// without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
// See the GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License along with this program.
// If not, see <https://www.gnu.org/licenses/>.

// NOTES:
// - A 'special_genres' option is supposed to be available through the MPD configuration or
//   the MPD client configuration, containing the list of musical genres for which the artist may
//   not correspond to the performer, such is for the 'Classical' genre (e.g. 'Classical, Opera,
//   Chamber Music etc.'). If this feature is not available then the procedure should be adapted
//   as indicated.
// - A 'various_equivs' option is supposed to be available through the MPD configuration or
//   the MPD client configuration, containing a list of synonyms or equivalent alternatives for
//   'Variuos' (e.g. 'Various artist, AA.VV., VVAA'). If this feature is not available then the
//   procedure should be adapted as indicated.
// - The CUE file being parsed is referenced here as <file.cue>.
// - The (MPD or client) music tag database is referenced here as 'musicdb[][]'.
// - The GETCUECOMMAND() subprocedure and the ISCONTAINED() function are defined at the bottom
//   of the main pseudocode.
// - Some subprocedures or functions are referenced (called) in this pseudocode although they are
//   not defined here (but just simply described).
// - The '&&' operator is intended for string concatenation.
// - Variables in this pseudocode are supposed to be craeted when set for the first time, being
//   initially empty (not defined or null); data types are not declared/defined.
// - Indexes of all the arrays in this pseudocode are supposed starting from 1.
// - Runtime exceptions are generally not handled here.
// - More details and code explanations are among the comments to the pseudocode.

// IMPORTANT WARNING:
// Please, to avoid uncoherent semantic results, **BE VERY CAREFUL** altering the "cascading" way
// parsing is done! Be aware that changes that may appear as possible optimizations of the code
// flow, could lead to assignments that lose the semantic connections between the tags.
//
// VERSION HISTORY
// - 2.0 (2019-05-08)
//      NOTE: This version superseeds all the previous ones, which are now deprecated.
//      + Code flow completely revised and partially rewrited.
//      + Code updated to take into account the changes in the new version 2.0 of the main document
//        (SUPPORTING CUE FILES IN MPD: SYNTACTIC AND SEMANTIC CONSISTENCY WITH STANDARD FORMATS FOR
//        AUDIO METADATA), including changes related to the referenced CUE tags.
//      + Function GETCUECOMMAND() rewrited.
//      + Fixed various bugs and errata coming from the previous version.
// - 1.3 (2018-11-15)
//      + Initial version of this pseudocode.

BEGIN

// initialize the cuelist[] array, containing the list of all possible "cleaned" CUE tags
SET cuelist[] = ('DISCID', 'ALBUMID', 'GENRE', 'DATE', 'ORIGINALDATE', 'ORIGINALYEAR',
                 'MEDIA', 'SOURCEMEDIA', 'DISC', 'DISCNUMBER', 'TOTALDISCS', 'DISCTOTAL',
                 'CATALOG', 'LABEL', 'COPYRIGHT', 'LOCATION',
                 'TITLE', 'PERFORMER', 'COMPOSER', 'SONGWRITER',
                 'FILE', 'TRACK', 'ISRC', 'PREGAP', 'INDEX', 'POSTGAP',
                 'REM', 'COMMENT')

// open the CUE file for reading
OPEN <file.cue> FOR READ
// read the CUE file line by line, storing CUE commands and respective tags values
SET lineno = 0  // initialize variable for CUE lines count
SET ntracks = 0 // initialize variable for CUE tracks count
SET headerlastlineno = -1 // initialize variable for store the last line number of the CUE header
WHILE NOT EOF <file.cue>
    READ cueline FROM <file.cue>
    IF (cueline IS EMPTY) THEN
        CONTINUE
    ENDIF
    // call the GETCUECOMMAND() function subprocedure to parse the current CUE line and store the
    // contained CUE command to a bi-dimensional indexed array cuecmd[][] where:
    // - the containing CUE command tag in the current cueline being assigned to cuecmd[lineno]->tag
    //   examples: 'TITLE', 'GENRE', 'COMPOSER' etc.
    // - the corresponding value (if not empty) being assigned to cuecmd[lineno]->val
    //   examples: 'Autobahn', 'Classical', 'Mozart' etc.
    // examples:
    //      'REM GENRE "Alternative Rock"' line parsing would give:
    //              cuecmd[lineno]->tag = 'GENRE'
    //              cuecmd[lineno]->val = 'Alternative Rock'
    //      'TRACK 03 AUDIO' line parsing would give:
    //              cuecmd[lineno]->tag = 'TRACK'
    //              cuecmd[lineno]->val = '03'
    //      'INDEX 01 01:17:43' line parsing would give:
    //              cuecmd[lineno]->tag = 'INDEX'
    //              cuecmd[lineno]->val = '01 01:17:43'
    IF (GETCUECOMMAND(cueline, cuelist[], lineno, cuecmd[][]) IS FALSE) THEN
        CONTINUE
    ENDIF
    // if the CUE command in the current line is 'FILE', it means that the previous one was the
    // last line of the CUE file disc section (indeed the tracks section begins at the line
    // carrying on the *first* 'FILE' CUE command)
    IF ((headerlastlineno IS -1) AND (cuecmd[lineno]->tag IS 'FILE')) THEN
        SET headerlastlineno = lineno - 1
    ENDIF
    // if the CUE command in the current line is 'TRACK', then increase the track counter
    IF (cuecmd[lineno]->tag IS 'TRACK') THEN
        SET ntracks = ntracks + 1
    ENDIF
    // increase the line counter before to read the next line
    SET lineno = lineno + 1
ENDWHILE
// once read close the CUE file
CLOSE <file.cue>

// initialize an array list of musical genres for which the artist may not correspond to the
// performer by getting items through a GETCONFOPT() function subprocedure
// NOTE: a 'special_genres' option is supposed to be available through the program configuration
SET specialgenres[] = GETCONFOPT('special_genres')  // NOTE: this subprocedure is not defined here
// initialize a flag to indicate if the album genre is in the specialgenres[] array list
SET isspecialgenre = FALSE

// initialize an array list of synonyms or equivalent alternatives for 'Variuos' by getting items
// through a GETCONFOPT() function subprocedure
// NOTE: a 'various_equivs' option is supposed to be available through the program configuration
SET variousequivs[] = GETCONFOPT('various_equivs')  // NOTE: this subprocedure is not defined here

// now parsing the CUE disc section...
// go down throught the cuecmd[][] array staying in the range of the CUE file disc section
// (from the 1st line down to the last line in the disc section)
FOR lineno = 1 TO headerlastlineno
    // store in a header[] array the values which will be need later when parsing the track section
    IF NOT (IS EMPTY cuecmd[lineno]->val) THEN
        CASE cuecmd[lineno]->tag IS
            'REM':
            'COMMENT':
                // note: there may be more instances of 'REM' or 'REM COMMENT'...
                SET header[comment] = header[comment] && ' ' && cuecmd[lineno]->val
                BREAK
            'GENRE':
                SET header[genre] = cuecmd[lineno]->val
                // check if the header[genre] value is contained in the specialgenres[] array list
                //  NOTE: if the 'special_genres' option is not supported then the next code line
                //        should be replaced by some appropriated code such as:
                //        IF (header[genre] IS 'Classical') THEN
                IF (ISCONTAINED(header[genre], specialgenres[]) IS TRUE) THEN
                    SET isspecialgenre = TRUE
                ENDIF
                BREAK
             'TITLE':
                SET header[album] = cuecmd[lineno]->val
                BREAK
            'PERFORMER':
                // in the disc section of a CUESHEET, 'PERFORMER' should contain the artist to
                // whom the entire album is referred (which is associated with 'albumartist' tag)
                SET header[albumartist] = cuecmd[lineno]->val
                // if header[albumartist] is in the variousequivs[] array list then use the standard
                // string for various artists ('Various')
                // NOTE: if the 'various_equivs option' is not supported then the whole condition
                //       in the next three code lines should be removed (or replaced with another
                //       appropriate check (something as: if albumartist starts with 'various'...)
                IF (ISCONTAINED(header[albumartist], variousequivs[]) IS TRUE) THEN
                    SET header[albumartist] = 'Various'
                ENDIF
                BREAK
            'COMPOSER':
                SET header[albumcomposer] = cuecmd[lineno]->val  // used later as a fallback
                BREAK
            // ... put here similar assignments for other standard or extended non-standard CUE
            //     commands which should be managed by the client:
            //     'DATE', 'CATALOG', 'SONGWRITER'
            //     'DISCID', 'ALBUMID', 'ORIGINALYEAR', 'ORIGINALDATE', 'ORIGINALYEAR',
            //     'MEDIA', 'SOURCEMEDIA', 'DISC', 'DISCNUMBER', 'TOTALDISCS', 'DISCTOTAL',
            //     'CATALOG', 'LABEL', 'COPYRIGHT', 'LOCATION',
        ENDCASE
    ENDIF
ENDFOR

// now parsing the CUE tracks section...
// continue go down throught the cuecmd[][] array but now starting from the top of the track section
SET trackno = 1  // inizialize variable for track number index
FOR lineno = headerlastlineno + 1 to nlines
    // if command is 'FILE':
    // get the current audio file and restart timecode positioning for its tracks
    IF (cuecmd[lineno]->tag IS 'FILE') THEN
        // call the SETFILE() subprocedure to parse cuecmd[lineno]->val and set the audio file path
        // and type (i.e. WAVE) in the MPD database
        CALL SETFILE(cuecmd[lineno]->val)  // NOTE: this subprocedure is not defined here
        // call the RESTARTPOSITIONTIME() subprocedure to reset the time position counts
        // for the tracks related to the current audio file to which they belong
        CALL RESTARTPOSITIONTIME()  // NOTE: this subprocedure is not defined here
        // no further action is needed now, so jump directly to read next line
        NEXTFOR
    ENDIF
    // if command is 'TRACK':
    IF (cuecmd[lineno]->tag IS 'TRACK') THEN
        // assign the track number as is specified in the CUE file
        SET musicdb[track[trackno]]->track = cuecmd[lineno]->val
        // assign tags taken from the disc section to the current track
        SET musicdb[track[trackno]]->genre = header[genre]
        SET musicdb[track[trackno]]->album = header[album]
        SET musicdb[track[trackno]]->albumartist = header[albumartist]
        SET musicdb[track[trackno]]->comment = header[comment]
        // ... put here similar assignments for all the other CUE commands as already stored
        //     in the header[] array, when parsing the CUE disc section before ...
        // now go ahead to the next track...
        SET trackno = trackno + 1
        // no further action is needed now, so jump to read next line
        NEXTFOR
    // if command is neither 'FILE' nor 'TRACK':
    ELSE
        // still being inside the current track, now set the remaining specific tags for the track...
        IF ( (cuecmd[lineno]->tag IS 'PREGAP') OR
             (cuecmd[lineno]->tag IS 'INDEX') OR
             (cuecmd[lineno]->tag IS 'POSTGAP') ) THEN
            // call the UPDATEPOSITIONTIME() subprocedure to update the timecode position
            // for the current track (trackno) inside the current audio file which it belongs
            CALL UPDATEPOSITIONTIME()   // NOTE: this subprocedure is not defined here
            // no further action is needed now, so jump to read next line
            NEXTFOR
        ENDIF
        CASE cuecmd[lineno]->tag IS
            'REM':
            'COMMENT':
                // note: append the track comments found with 'REM' or 'REM COMMENT'...
                IF (cuecmd[lineno]->val IS NOT EMPTY) THEN
                    SET musicdb[track[trackno]]->comment =
                        musicdb[track[trackno]]->comment && ' ' && cuecmd[lineno]->val
                ENDIF
                BREAK
            'TITLE':
                IF (cuecmd[lineno]->val IS NOT EMPTY) THEN
                    SET musicdb[track[trackno]]->title = cuecmd[lineno]->val
                ENDIF
                BREAK
            'PERFORMER':
                IF (cuecmd[lineno]->val IS NOT EMPTY) THEN
                    SET musicdb[track[trackno]]->performer = cuecmd[lineno]->val
                ENDIF
                BREAK
            'COMPOSER':
                IF (cuecmd[lineno]->val IS NOT EMPTY) THEN
                    SET musicdb[track[trackno]]->composer = cuecmd[lineno]->val
                ENDIF
                BREAK
            // ... put here similar cases for assign the remaining specific tags for the
            //     track: 'ISRC', 'SONGWRITER'
        ENDCASE
        // now set artist...
        //  NOTE: if the 'special_genres' option is not supported then the next code line should be
        //        replaced by some appropriated code such as:
        //        IF (musicdb[track[trackno]]->genre IS 'Classical') THEN
        IF (isspecialgenre IS TRUE) THEN
        // artist when genre IS in the 'special_genres' list:
            IF (musicdb[track[trackno]]->composer IS NOT EMPTY) THEN
                SET musicdb[track[trackno]]->artist = musicdb[track[trackno]]->composer
            ENDIF
        ELSE
        // artist when genre IS NOT in the 'special_genres' list:
            IF (musicdb[track[trackno]]->performer IS NOT EMPTY) THEN
                SET musicdb[track[trackno]]->artist = musicdb[track[trackno]]->performer
            ELSE
                IF (musicdb[track[trackno]]->albumartist IS NOT 'Various') THEN
```

```
                    SET musicdb[track[trackno]]->artist = musicdb[track[trackno]]->albumartist
                ENDIF
            ENDIF
        ENDIF
        // fallbacks for artist, performer and composer...
        IF ((musicdb[track[trackno]]->artist IS EMPTY) AND
            (ISCONTAINED(musicdb[track[trackno]]->albumartist, variousequivs[]) IS FALSE)) THEN
            SET musicdb[track[trackno]]->artist = musicdb[track[trackno]]->albumartist
        ENDIF
        IF ((musicdb[track[trackno]]->performer IS EMPTY) AND
            (musicdb[track[trackno]]->artist IS NOT EMPTY) THEN
                SET musicdb[track[trackno]]->performer = musicdb[track[trackno]]->artist
        ENDIF
        IF ((musicdb[track[trackno]]->composer IS EMPTY) AND
            (musicdb[track[trackno]]->albumcomposer IS NOT EMPTY) THEN
                SET musicdb[track[trackno]]->composer = musicdb[track[trackno]]->albumcomposer
        ENDIF
        // final fallback assignments...
        // NOTE: for genres, see: Sony CDTEXT genres list; for album, albumartist, artist and title,
        //        'Unknown' is as used in freedb, MusicBrainz etc.
        IF (IS EMPTY musicdb[track[trackno]]->genre) THEN SET musicdb[track[trackno]]->genre = 'Not Defined'
        IF (IS EMPTY musicdb[track[trackno]]->album) THEN SET musicdb[track[trackno]]->album = 'Unknown'
        IF (IS EMPTY musicdb[track[trackno]]->albumartist) THEN SET musicdb[track[trackno]]->albumartist = 'Unknown
'
        IF (IS EMPTY musicdb[track[trackno]]->title) THEN SET musicdb[track[trackno]]->title = 'Unknown'
        IF (IS EMPTY musicdb[track[trackno]]->artist) THEN SET musicdb[track[trackno]]->artist = 'Unknown'
        IF (IS EMPTY musicdb[track[trackno]]->performer) THEN SET musicdb[track[trackno]]->performer = 'Unknown'
        IF (IS EMPTY musicdb[track[trackno]]->composer) THEN SET musicdb[track[trackno]]->composer = 'Unknown'
    ENDIF
ENDFOR

// Function GETCUECOMMAND(cueline, cuelist[], lineno, cuecmd[][])
//      parse a CUE line (cueline) corresponding to the lineno-th line of the CUE file (lineno),
//      looking for a CUE command (listed in cuelist[]) and store the contained CUE command tag and
//      its value to the corresponding element (cuecmd[lineno][]) in the cuecmd[][] array
// NOTES:
//      Return: TRUE if a valid CUE command is found; FALSE if NOT found.
// NOTES:
//      - the REGEXMATCH(string, test, options) is intended to be a function where:
//          + string: input text to be matched
//          + test: matching regexp
//          + options: test matching options
//          and the return value is:
//          + NULL, if test DOES NOT match string
//          + a ordered vector containing the matches found, if test DOES match string
//      - the REGEXREPLACE(string, test, options, replace) is intended to be a function where:
//          + string: input text to be matched
//          + test: matching regexp
//          + options: test matching options
//          + replace: substitution regexp
//          and the return value is:
//          + NULL, if test DOES NOT match string
//          + the result of the regexp match and substitution, if test DOES match string
//          + the REGEXMATCH() and REGEXREPLACE() functions are intended to be provided by the
//            programming language actually used in the implementation
//
FUNCTION GETCUECOMMAND(cueline, cuelist[], lineno, cuecmd[][])
    // set the regexp test and test options
    SET test = '/^\s*(REM){0,1}\s*([a-zA-Z]*){0,1}\s*(.*)$/'
    SET options = 'is'
    // now do the regexp matching calling a REGEXMATCH function with:
    //   - option 'i' (case insensitive, non global)
    //   - option 's' (enables the dot (.) metacharacter to also match '\n' new lines)
    // REGEXMATCH should return a matches[] vector storing the found matches
    // NOTE: the test would return 3 matching items:
    //       - the 1st item contains 'REM' if present (if not present it is empty)
    //       - the 2nd item contains the CUE command OR will be empty when the matching
    //         string is a standard 'REM comment' CUE command
    //       - the 3rd item contains the remaining part in the line
    // for example:
    //   when:       cueline=' REM COMPOSER "Mozart"   '
    //   the test will give:
    //       - 1st item = 'REM' (without trailing and leading white spaces)
    //       - 2nd item = 'COMPOSER' (without trailing and leading white spaces)
    //       - 3rd item = '"MOZART"   'Mozart" (with final white spaces)
    SET matches[] = REGEXMATCH(cueline, test, options)
    // now check and set the CUE command
    IF ((matches[2] IS EMPTY) AND (matches[1] == 'REM')) THEN
        // is the standard 'REM comment'...
        SET cuecmd[lineno]->tag = matches[1]
        SET cuecmd[lineno]->val = matches[3]
    ELSEIF ((matches[2] IS NOT EMPTY) AND (ISCONTAINED(matches[2], cuelist[]))) THEN
        // is a valid standatd CUE command (but not REM)...
        SET cuecmd[lineno]->tag = matches[2]
        // before set the value, strip all the leading and final white spaces and tabs and also the
        // quotation marks ("): do a regexp match and substitution calling a REGEXREPLACE function
        // with the global ('g') option, matching white spaces and substitute with NULL
        SET test = '/^\s*\"{0,1}\s*|\s*\"{0,1}\s*$/'
        SET options = 'g'
        SET replace = NULL
        SET cuecmd[lineno]->val = REGEXREPLACE(matches[3], test, options, replace)
    ELSE
        // otherwise something wrong has happened...
        RETURN FALSE
    ENDIF
    // a valid CUE command was found...
    RETURN TRUE
END PROCEDURE

// Function ISCONTAINED(item, itemsarray[])
//      check if the item value matches the value of one of the elements in the itemsarray[]
// NOTES:
//      Return: TRUE if a match is found; FALSE if a match is NOT found.
//
FUNCTION ISCONTAINED(item, itemsarray[])
    FOR i IN itemsarray[]
        IF (item IS itemsarray[i]) THEN RETURN TRUE
    ENDFOR
    RETURN FALSE
END FUNCTION

END
// end of cuesheet-parsing pseudocode
```

6.2 EXPECTED RESULTS

The following are some synthetic examples related to some typical situations, which are provided here to show the expected results with this implementation.

– **Generic CUE file**:

```
REM GENRE 'Rock Music'
TITLE 'The Dark Side Of The Moon'
PERFORMER 'Pink Floyd'
...
TRACK 02 AUDIO
  TITLE 'On The Run'
  PERFORMER 'Pink Floyd'
  REM COMPOSER 'Gilmour, Waters'
...
```

… and the correspondig tags in the music tag database, as they should result by the proposed pseudocode:

- *track*: 02
- *genre*: Rock Music
- *album*: The Dark Side Of The Moon
- *albumartist*: Pink Floyd
- *title*: On The Run
- *artist*: Pink Floyd
- *performer*: Pink Floyd
- *composer*: Gilmour, Waters

– **A CUE file with a genre for which the artist may not correspond to the performer**:

```
REM GENRE 'Classical'
TITLE 'Requiem'
PERFORMER 'Gabriel Fauré'
...
TRACK 02 AUDIO
  TITLE 'Offertoire'
  PERFORMER 'Academy of St Martin in the Fields'
  REM COMPOSER 'Gabriel Fauré'
...
```

… and the correspondig tags in the music tag database, as they should result by the proposed pseudocode:

- *track*: 02
- *genre*: Classical
- *album*: Requiem
- *albumartist*: Gabriel Fauré
- *title*: Offertoire
- *artist*: Gabriel Fauré
- *performer*: Academy of St Martin in the Fields
- *composer*: Gabriel Fauré

– **A CUE file for a multi-artist album**:

```
REM GENRE 'Alternative Rock'
TITLE 'Alternative Hits of 2000s'
PERFORMER 'Various'
...
TRACK 14 AUDIO
  TITLE 'This World Hell'
  PERFORMER 'Killing Joke'
  REM COMPOSER 'Killing Joke'
...
```

… and the correspondig tags in the music tag database, as they should result by the proposed pseudocode:

- *track*: 14
- *genre*: Alternative Rock
- *album*: Alternative Hits of 2000s
- *albumartist*: Various
- *title*: This World Hell
- *artist*: Killing Joke
- *performer*: Killing Joke
- *composer*: Killing Joke

– **A CUE file for a classical music multi-artist album**:

```
REM GENRE 'Classical'
TITLE 'The Best Piano Music'
PERFORMER 'Various'
...
TRACK 07 AUDIO
  TITLE 'Nocturne for piano No. 13'
  PERFORMER 'Maria João Pires'
  REM COMPOSER 'Frédéric Chopin'
...
```

… and the correspondig tags in the music tag database, as they should result by the proposed pseudocode:

- *track*: 07
- *genre*: Classical
- *album*: The Best Piano Music
- *albumartist*: Various
- *title*: Nocturne for piano No. 13
- *artist*: Frédéric Chopin
- *performer*: Maria João Pires
- *composer*: Frédéric Chopin

7 CONCLUSIONS

Support for CUE files in MPD and its clients is not as trivial and simple as it might appear when when trying to ensure syntactic and semantic consistency between the involved formats for audio metadata.

With the aim of overcoming the critical issues, the specifications of the main formats were first examined: CCDB (freedb and MusicBrainz), TOC/CD-TEXT, CUESHEET, Vorbis comment and, obviously, MPD tags.

Having considered the results of the analisys carried out on these formats, it was possible to deepen the syntactic and semantic knowledge of the CUESHEET format, also in relation to the use conventions generally adopted in CUE files.

Then, the correlations between CUE and MPD metadata were discussed, identifying a common set of audio metadata to be supported, exactly specifying their meaning and use, and finally understanding how to establish correlations so as to ensure syntactic and semantic consistency.

Also, this has meant to *precisely specify the syntactic and semantic use of the involved (standard and non-standard) CUE commands*, and to *define a proposal to extend the MPD specification* introducing several new extra *“extended”* tags, that were necessary to achieve the optimal correlations.

All of this has been reported in a *reference summary for CUE and MPD audio metadata* (both standard and extended), which also contains the proposals to standardize and extend both the CUE commands and the MPD tags.

Finally, a *pseudocode* has been proposed, which implements the parsing of CUE file commands in relation to the MPD tags so as to realize the appropriate correlations between the medatates. Some synthetic examples showing what result should be obtained with this implementation were also given.

The hope is that the results and proposals of this work could be useful for improving CUE files support in MPD and its clients, so allowing users to better manage and use their CUE files to organize and listen to their digital audio collections.

8 VERSION HISTORY

- ***2.0 (2019-05-08)***
  NOTE: ***This version superseed any previous one, which now are deprecated***.
  - Document generally revised and partially rewrited, also to take into account the changes listed below.
  - PSEUDOCODE source code and logical flow completely revised and partially rewrited, also to take into account the changes listed below. Fixed various bugs and errata coming from the previous version.
  - Added `label` to the MPD's tag list (which by mistake had not been listed before).
  - Added the new MPD (>=0.21) `originaldate` tag.
  - Added CUESHEET correspondences to `label` (`REM LABEL`) and `originaldate`.
  - Added `REM COMMENT` as a further CUESHEET correspondence to `comment`.
  - "REFERENCE SUMMARY FOR CUESHEET…" revised and updated in accordance with the above.
  - Added chapter "CUESHEET EXAMPLE".
  - General syntax and typo revision.
- ***1.3 (2018-11-15)***
  - Fixed some *errata* in the pseudocode (missing `COMPOSER` and `ORIGINALDATE` in `cuelist[]` initialization).
- ***1.2 (2018-11-11)***
  - Fixed some *errata* in the reference summary table.
  - Summary chapter partially revised.
- ***1.1 (2018-11-10)***
  - Typo revision.
- ***1.0 (2018-11-09)***
  - Initial version.

---

1. CDDB < https://en.wikipedia.org/wiki/CD_database > ↩
2. CDDA < https://en.wikipedia.org/wiki/Compact_Disc_Digital_Audio > ↩
3. FreeDB < https://en.wikipedia.org/wiki/Freedb > ↩
4. FREEDB FILE FORMAT < http://ftp.freedb.org/pub/freedb/latest/DBFORMAT > ↩
5. FREEDB database format < http://ftp.freedb.org/pub/freedb/misc/freedb_database_format_specs.zip > (zip file download) ↩
6. FREEDB HowTo < http://ftp.freedb.org/pub/freedb/misc/freedb_howto1.07.zip > (zip file download) ↩
7. See: Audio tracks < https://en.wikipedia.org/wiki/Track_(optical_disc)#Audio_tracks > ↩
8. MusicBrainz < https://en.wikipedia.org/wiki/MusicBrainz > ↩
9. MusicBrainz Picard < https://en.wikipedia.org/wiki/MusicBrainz_Picard > ↩
10. ID3v2 documents < https://id3.org/Developer%20Information > ↩
11. MusicBrainz Database < https://musicbrainz.org/doc/MusicBrainz_Database > ↩
12. MusicBrainz Database schema < https://musicbrainz.org/doc/MusicBrainz_Database/Schema > ↩
13. MusicBrainz Picard tags < https://picard.musicbrainz.org/docs/tags/ > ↩
14. MusicBrainz Picard tag mappings < https://picard.musicbrainz.org/docs/mappings/ > ↩
15. MusicBrainz Picard Genre List < https://picard.musicbrainz.org/docs/mappings/ > ↩
16. CDDA < https://en.wikipedia.org/wiki/Compact_Disc_Digital_Audio > ↩
17. Red Book < https://en.wikipedia.org/wiki/Rainbow_Books#Red_Book_(1980) > ↩
18. Rainbow_Books < https://en.wikipedia.org/wiki/Rainbow_Books > ↩
19. See details in the TOC FILES section of the Cdrdao manpage < https://manpages.debian.org/stretch/cdrdao/cdrdao.1.en.html > ↩
20. CD-TEXT < https://en.wikipedia.org/wiki/CD-Text > ↩
21. CDDA < https://en.wikipedia.org/wiki/Compact_Disc_Digital_Audio > ↩
22. CD-TEXT Format < https://www.gnu.org/software/libcdio/cd-text-format.html > ↩
23. MMC-3 (Rev.10g, Nov. 2001) < http://www.13thmonkey.org/documentation/SCSI/mmc3r10g.pdf > ↩
24. Red Book < https://en.wikipedia.org/wiki/Rainbow_Books#Red_Book_(1980) > ↩
25. Rainbow_Books < https://en.wikipedia.org/wiki/Rainbow_Books > ↩
26. MMC-3 (Rev.10g, Nov. 2001) < http://www.13thmonkey.org/documentation/SCSI/mmc3r10g.pdf > ↩
27. CUE file < https://en.wikipedia.org/wiki/Cue_file > ↩
28. CDRWIN < https://en.wikipedia.org/wiki/CDRWIN > ↩
29. CDRWIN User's Guide (goldenhawk.com) < https://web.archive.org/web/20070221154246/http://www.goldenhawk.com/download/cdrwin.pdf > ↩
30. CDRWIN User's Guide (digitalx.org) < https://web.archive.org/web/20151028040029/http://web.archive.org:80/web/20070221154246/http://www.goldenhawk.com/download/cdrwin.pdf > ↩
31. CUE sheet syntax (digitalx.org) < https://web.archive.org/web/20160201021136/http://digitalx.org/cue-sheet/syntax/ > ↩
32. CUE sheet info (hydrogenaud.io) < http://wiki.hydrogenaud.io/index.php?title=Cue_sheet > ↩
33. International Article Number < https://en.wikipedia.org/wiki/International_Article_Number > i.e. MCN ↩
34. CUE sheet syntax (digitalx.org) < https://web.archive.org/web/20160201021136/http://digitalx.org/cue-sheet/syntax/ > ↩
35. International Standard Recording Code < https://en.wikipedia.org/wiki/International_Standard_Recording_Code > ↩
36. CUE sheet syntax (digitalx.org) < https://web.archive.org/web/20160201021136/http://digitalx.org/cue-sheet/syntax/ > ↩
37. Non-compliant CUE sheets (hydrogenaud.io) < http://wiki.hydrogenaud.io/index.php?title=Cue_sheet#Multiple_files_with_gaps_.28Noncompliant.29 > ↩

38. EAC malformed cuesheets < https://web.archive.org/web/20110718230838/http://www.digital-inn.de/exact-audio-copy-english/40013-why-i-hate-eac-malformed-cue-sheets-4.html?s=3ce06844aaaadbf3cdf651003ac6c7bf#post148362 > ↩
39. Vorbis comment < https://en.wikipedia.org/wiki/Vorbis_comment > ↩
40. VorbisComment (XiphWiki) < https://wiki.xiph.org/VorbisComment > ↩
41. Ogg Vorbis Documentation (xiph.org) < https://www.xiph.org/vorbis/doc/v-comment.html > ↩
42. Ogg Vorbis Proposed field names < https://wiki.xiph.org/Field_names > ↩
43. Proposals for extending Ogg Vorbis comments < https://web.archive.org/web/20120429102447/http://reallylongword.org/vorbiscomment/ > ↩
44. Ogg Vorbis Documentation (xiph.org) < https://www.xiph.org/vorbis/doc/v-comment.html > ↩
45. Ogg Vorbis Proposed field names < https://wiki.xiph.org/Field_names > ↩
46. MPD Music Player < https://en.wikipedia.org/wiki/MPD_(Music_Player) > ↩
47. MPD protocol < https://www.musicpd.org/doc/protocol/index.html > ↩
48. MPD tags < https://www.musicpd.org/doc/protocol/tags.html > ↩
49. CDRWIN User’s Guide (goldenhawk.com) < https://web.archive.org/web/20070221154246/http://www.goldenhawk.com/download/cdrwin.pdf > ↩
50. CDRWIN User’s Guide (digitalx.org) < https://web.archive.org/web/20151028040029/http://web.archive.org:80/web/20070221154246/http://www.goldenhawk.com/download/cdrwin.pdf > ↩
51. CUE sheet syntax (digitalx.org) < https://web.archive.org/web/20160201021136/http://digitalx.org/cue-sheet/syntax/ > ↩
52. CD-TEXT Format: musical genres < https://www.gnu.org/software/libcdio/cd-text-format.html#table_003agenres > ↩
53. CD-TEXT Format: musical genres < https://www.gnu.org/software/libcdio/cd-text-format.html#table_003agenres > ↩