MusicBrainz Picard

Release v2.3.2

Jun 20, 2020

CONTENTS

INTRODUCTION

MusicBrainz Picard is a cross-platform music file tagger. This User Guide is intended to augment the information provided on the Picard website[1] and to provide an alternate format for the documentation, including a PDF document suitable for printing. Links to additional information such as scripts, plugins and tutorials are provided when available rather than trying to reproduce the information in this document.

In order to effectively use Picard, it is important to understand what the program can do and, equally important, what it cannot do. What Picard does, it does very well, but if you're expecting it to automatically organize your collection of thousands of random music files you will likely be disappointed. To quote from the Picard website, *"Picard is not built to be a mass single-track tag fixer. Picard believes in quality over quantity and provides a plethora of customizations to tweak music collections to your needs."*

1.1 Documentation Credits & Updates

This document only exists because of the volunteer effort that went into its development, from the initial documentation on the Picard website, the information posted in the Community Discussion Forum, documentation from scripts, plugins and program source code, proofreaders, editors, translators, and feedback from the user community. Individual contributors are not being listed here for fear of missing someone; however, please know that your contributions are greatly appreciated.

If you notice an error in the documentation or have additional material to contribute, please raise an issue[2] on the documentation project on GitHub[3]. Pull Requests[4] to address outstanding issues are also appreciated.

Bob Swift (rdswift)

Editor

---

[1] https://picard.musicbrainz.org/
[2] https://github.com/rdswift/picard-docs/issues
[3] https://github.com/rdswift/picard-docs/
[4] https://github.com/rdswift/picard-docs/pulls

1.2 Picard Can. . .

. . . add metadata tags to your music files, based on information available from the MusicBrainz website[5].

. . . look up the metadata either manually or automatically based on existing information, including artist and song name, disc id (for CDs), and a track's AcoustID fingerprint.

. . . retrieve and embed coverart images from a variety of sources.

. . . rename and place the music files in directories based on naming template instructions provided in a naming script.

. . . calculate and submit a disc id to the MusicBrainz database, attaching it to a specified release.

. . . calculate and submit a music file's AcoustID fingerprint to the AcoustID database[6].

1.3 Picard Cannot. . .

. . . automatically identify and remove all duplicate music files in your collection.

. . . provide metadata not already existing in the MusicBrainz database.

1.4 Limitations

File Formats

Picard currently supports most music file formats, with two notable exceptions: Matroska (.mka) and Windows WAVE (.wav). WAV files cannot be tagged due to the lack of a standard for doing so, however, they can be fingerprinted and renamed. In addition, Picard does not support writing custom tags for all formats. The Picard Tag Mappings[7] webpage provides more information regarding the mapping between Picard internal tag names and various tagging formats.

Request Rate Limiting

Picard's metadata retrieval is limited to the standard **one request per second** rate limiting for the MusicBrainz API. This becomes quite noticable when trying to process a large list of files, and is exacerbated by extensions that perform additional information requests from the database.

Network File Processing

Sometimes Picard needs to rewrite the entire music file in order to add or update the tags. This can take a few seconds, and the delay becomes even longer if the file is accessed across a network (e.g.: file is read from or written to a NAS device). The recommended "best practice" is to process all files on a local drive and then move them to the desired remote directory once processing is complete.

---

[5] https://musicbrainz.org/
[6] https://acoustid.org/
[7] https://picard.musicbrainz.org/docs/mappings/

GLOSSARY OF TERMS

Many of the terms used in this documentation and within Picard itself have specific meaning in the Music-Brainz environment. Specific terms are defined as follows:

acoustic fingerprint

> An acoustic fingerprint is a digital summary of an audio signal, that can be used to quickly identify the audio.
>
> Please see Wikipedia[8] for a full explanation of acoustic fingerprinting.

AcoustID

> AcoustID is an acoustic fingerprint system built entirely on open-source technology. See the AcoustID website[9] for additional information.

albumartist

> The musician or group of musicians performing on a release. For example, "The Beatles" is the albumartist for the album "Past Masters, Volume One[10]", while the albumartist for "No Boundaries: A Benefit for the Kosovar Refugees[11]" is "Various Artists".
>
> ---
>
> **Note:** The albumartist usage is different for Classical Music releases, which follow the Music-Brainz Classical Style Guide[12], listing the composer(s) first, followed by the performers.
>
> ---

artist

> The musician or group of musicians performing on a track. For example, "Jeen" is the artist on the track "Be (One in a Million)[13]" on the album "Tourist[14]".
>
> Please see the Artist[15] page on the MusicBrainz website for additional information.

---

[8] https://wikipedia.org/wiki/Acoustic_fingerprint

[9] https://acoustid.org/

[10] https://musicbrainz.org/release/9383a6f5-9607-4a36-9c68-8663aad3592b

[11] https://musicbrainz.org/release/65536c6a-9219-4c41-9829-781eab7cdb50

[12] https://musicbrainz.org/doc/Style/Classical

[13] https://musicbrainz.org/track/5acda7a7-697c-4614-8467-7c48b3d946a6

[14] https://musicbrainz.org/release/472f4da8-c7dd-4e4a-8aae-9e7824f85afc

[15] https://musicbrainz.org/doc/Artist

Note: The artist usage is different for Classical Music releases, which follow the MusicBrainz Classical Style Guide, showing only the composer and not the performers.

---

artist credit

Artist credits indicate who is the main credited artist (or artists) for releases, release groups, tracks and recordings, and how they are credited. They consist of artists, with (optionally) their names as credited in the specific release, track, etc., and join phrases between them. For example, on the release "Love Sponge[16]" the artist is "Walk off the Earth[17]" but is credited as "Gianni and Sarah".

Please see the Artist Credits[18] page on the MusicBrainz website for additional information.

caa

The Cover Art Archive[19] which is a joint project between the Internet Archive[20] and MusicBrainz[21], whose goal is to make cover art images available to everyone on the Internet in an organised and convenient way.

Please see the Cover Art Archive page[22] on the MusicBrainz website for additional information.

mbid

The MusicBrainz Identifier, which is a unique code used to identify each element in the MusicBrainz database. These are 128-bit Universally Unique Identifiers (UUID) represented as 32 hexadecimal digits, displayed in five groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters.

Please see the UUID page on Wikipedia[23] for more information.

non-album track

This term is obsolete and has been replaced with 'standalone recording'.

recording

An entity in MusicBrainz which can be linked to tracks on releases. For example "Be (One in a Million)" (recording id: fbd0e1a0-b6c5-4926-b89d-3922e4144da9) is the recording linked to track 14 on the album "Tourist". Each track must always be associated with a single recording, but a recording can be linked to any number of tracks.

Please see the Recording[24] page on the MusicBrainz website for additional information.

release

---

[16] https://musicbrainz.org/release/6ca797fd-8f3a-4326-bdc7-f805cb2de088
[17] https://musicbrainz.org/artist/e2a5eaeb-7de7-4ffe-a519-e18e427a5060
[18] https://musicbrainz.org/doc/Artist_Credits
[19] https://coverartarchive.org/
[20] https://archive.org/
[21] https://musicbrainz.org/
[22] https://musicbrainz.org/doc/Cover_Art_Archive
[23] https://en.wikipedia.org/wiki/Universally_unique_identifier
[24] https://musicbrainz.org/doc/Recording

Represents the unique issuing of a product on a specific date with specific release information such as the country, label, barcode and packaging. For example "Sea of No Cares[25]" (release id: 4e4ba41e-24ae-3f57-87f6-3d8f19ae9483) is one version of the album released by Great Big Sea.

Please see the Release[26] page on the MusicBrainz website for additional information.

release group

Groups several different releases into a single logical entity. Every release belongs to one, and only one release group. Both release groups and releases are "albums" in a general sense, but with an important difference: a release is something you can buy as media such as a CD or a vinyl record, while a release group embraces the overall concept of an album — it doesn't matter how many CDs or editions / versions it had. For example "Sea of No Cares" (release id: 4e4ba41e-24ae-3f57-87f6-3d8f19ae9483) is one version of the album in the release group id 7e7ffd2b-3d1b-3487-aaaf-e4e6037f09ca[27] by Great Big Sea.

Please see the Release Group[28] page on the MusicBrainz website for additional information.

standalone recording

A recording that is not linked to any release. An example is "Sea of No Cares (live)[29]" (recording id: 0198c132-ed38-430c-92bd-d3c7e9ff25b8) by Great Big Sea.

Please see the Standalone Recording[30] page on the MusicBrainz website for additional information.

track

A track is the way a recording is represented on a particular release (or, more precisely, on a particular medium). Every track has a title and is credited to one or more artists. For example, track 7 of the album "Back to Boston[31]" by Jason Anderson is "Driving Home[32]" (track id: bf8ecb3c-6fe6-41b7-a078-5748265a9f94).

Please see the Track[33] page on the MusicBrainz website for additional information.

work

A distinct intellectual or artistic creation, which can be expressed in the form of one or more audio recordings. While a 'Work' in MusicBrainz is usually musical in nature, it is not necessarily so. For example, a work could be a novel, play, poem or essay, later recorded as an oratory or audiobook. "Blinded by the Light[34]" by "Manfred Mann's Earth Band" is a recording of the work id 7a757d97-da2a-3751-8d32-94d471de2eeb[35] "Blinded by the Light" written by Bruce Springsteen.

---

[25] https://musicbrainz.org/release/4e4ba41e-24ae-3f57-87f6-3d8f19ae9483
[26] https://musicbrainz.org/doc/Release
[27] https://musicbrainz.org/release-group/7e7ffd2b-3d1b-3487-aaaf-e4e6037f09ca
[28] https://musicbrainz.org/doc/Release_Group
[29] https://musicbrainz.org/recording/0198c132-ed38-430c-92bd-d3c7e9ff25b8
[30] https://musicbrainz.org/doc/Standalone_Recording
[31] https://musicbrainz.org/release/9780e88d-a9e2-4e99-87c4-e54b65e7e49b
[32] https://musicbrainz.org/track/bf8ecb3c-6fe6-41b7-a078-5748265a9f94
[33] https://musicbrainz.org/doc/Track
[34] https://musicbrainz.org/recording/431b3d53-2783-46fd-9bb4-e1410f2941b6
[35] https://musicbrainz.org/work/7a757d97-da2a-3751-8d32-94d471de2eeb

Please see the Work[36] page on the MusicBrainz website for additional information.

See also:

For more information on these and other terms used, please refer to the Terminology[37] page on the Music-Brainz website.

See also:

For a detailed explanation of how all the elements are related within the MusicBrainz environment, please refer to the MusicBrainz Database / Schema[38] webpage.

---

[36] https://musicbrainz.org/doc/Work
[37] https://musicbrainz.org/doc/Terminology
[38] https://musicbrainz.org/doc/MusicBrainz_Database/Schema

GETTING STARTED

3.1 Download & Install

The latest version of MusicBraiz Picard is always available for download from the Picard Website[39]. This includes installers for all major operating systems (e.g.: Linux, macOS and Windows) as well as the source code[40]. The latest source code is also available at the project repository[41] on GitHub.

3.1.1 Installing Using Flatpak

Picard is also available on Flathub. This version should work on all modern Linux distributions, as long Flatpak is installed.

First enable the Flathub repository:

```
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.
↪flatpakrepo
```

You can now install Picard:

```
flatpak install flathub org.musicbrainz.Picard
```

---

[39] https://picard.musicbrainz.org/downloads/
[40] https://picard.musicbrainz.org/downloads/#source
[41] https://github.com/musicbrainz/picard

3.2 Main Screen
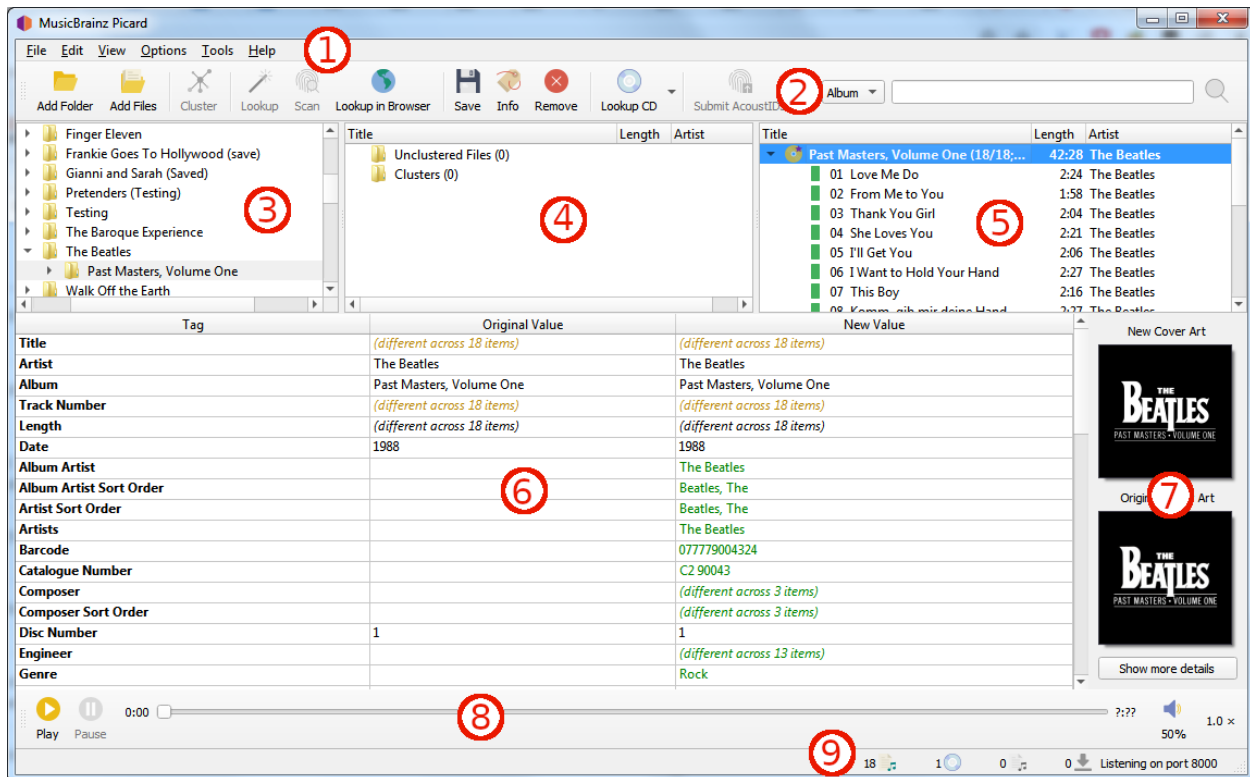
1. **Menu Bar**: This provides the pull-down menu of actions that Picard can perform.

2. **Tool Bar**: This provides quick links to the main functions performed by Picard. This can be customized by the user in the *User Interface Options* settings.

3. **File Browser**: This provides a browser for selecting files and directories for processing.

4. **Cluster Pane**: Often referred to as the "left-hand pane", this section allows the user to select and cluster files for scanning, lookup or matching.

5. **Album Pane**: Often referred to as the "right-hand pane", this section displays the albums retrieved from MusicBrainz. This is the section where files are matched to downloaded track information.

6. **Metadata Pane**: This section is a three-column table of the tag metadata for the album or track currently selected in the Album Pane. The first column shows the tag name, the second shows the original value found in the file, and the third column shows the new value that will be written.

7. **Cover Art**: This shows the new cover art image that will be written to the selected album or track, along with the original cover art image found in the files matched to the selected album or track.

8. **Player**: The built-in player that can be used to play selected audio files.

9. **Status Bar**: The bar at the bottom of the screen shows information about the current operation of Picard, including such items as number of files, albums, and pending downloads.

3.3 Status Icons

When albums and tracks are displayed in the right-hand pane, each line begins with an icon to indicate the status of the item.

3.3.1 Album / Release Icons

This icon indicates that the information for the release has been successfully retrieved from the MusicBrainz database. Some, but not all, tracks may have been matched to files and the information has not been modified.

This icon indicates that some of the tracks have been matched and that the information for the release has been modified.

This icon indicates that all of the tracks have been matched and that the information has not been modified.

This icon indicates that all of the tracks have been matched and that the information for the release has been modified.

This icon indicates that Picard has encountered an error with the release, typically while retrieving the information from the MusicBrainz database.

3.3.2 Track Icons

This icon indicates that the track is an audio track and that there is no single file currently matched. This appears if there is no file matched, of if there are multiple files matched.

This icon indicates that the track is a video track and that there is no file currently matched. This appears if there is no file matched, of if there are multiple files matched.

This icon indicates that the track is a data track and that there is no file currently matched. This appears if there is no file matched, of if there are multiple files matched.

These icons indicates the quality of match between the information from the file and the information for the track as provided from the MusicBrainz database. Red indicates a poor match, progressing to all green which indicates a very good match.

This icon indicates that the track has been saved successfully.

This icon indicates that Picard encountered an error while trying to save the track. This is typically due to the file being marked as read-only, or you do not have sufficient permission to save the file in the specified directory.

CHAPTER

FOUR

CONFIGURATION

Once Picard has been installed on your system, the next step is to configure it to your preferences. The configuration consists of enabling the desired screen sections for display, selecting the desired actions, and setting the various options.

4.1 Screen Setup

The screen setup is found under the *"View"* item on the menu bar. To enable the display of an item, simply check the box for the screen option. The items are:

File Browser

> This displays a file browser on the left side of the screen for selecting files and directories for processing. Files and directories can also be selected using your system's file browser by dragging and dropping them onto the Picard application.

Cover Art

> This displays the cover art for the currently selected item (track or release) in a window to the right of the tags section of the display. This allows you to select or replace the cover art saved with the release.

Actions

> This displays the button bar of the actions performed by Picard, located just below the menu bar.

Search

> This displays the manual search box to the right of the "Actions" button bar.

Player

> This displays the built-in player for playing selected audio files.

4.2 Action Options

The action options are found under the *"Options"* item on the menu bar. There are three available actions that Picard can perform when saving selected music files:

Rename Files

Picard will rename each file in accordance with the naming script.

Move Files

Picard will move files to the target directory in accordance with the naming script.

Save Tags

Picard will update the metadata tags in the files in accordance with the specified option settings and tagging scripts.

4.3 Option Settings

The option settings are found under the *"Options → Options. . . "* item on the menu bar. This will open a new window with the option groups listed in a tree format on the left hand side, and the individual settings on the right hand side. This is where the majority of Picard's customization is performed.

4.3.1 General Options

Server address

> The domain name for the MusicBrainz database server used by Picard to get details of your music. Default value: musicbrainz.org (for the main MusicBrainz server).

Port

> The port number for the server. Default value: 80 (for the main MusicBrainz server).

Username

> Your MusicBrainz website username, used to submit acoustic fingerprints, retrieve and save items to your collections, and retrieve personal folksonomy tags.

Password

> Your MusicBrainz website password.

Automatically scan all new files

> Check this box if you want Picard to scan each music file you add and look for an AcoustID fingerprint. This takes time, but may be helpful for you and MusicBrainz. Leave it unchecked

if you don't want Picard to do this scan automatically. In any case, you can direct Picard to scan a particular music file at any time using *"Tools → Scan"*.

Ignore MBIDs when loading new files

If you disable this option Picard will not use MusicBrainz identifiers (MBIDs) stored in the files to automatically load the corresponding MusicBrainz release and match the loaded file to the correct track. This is useful when re-processing files that have been previously tagged with incorrect information.

Check for updates during start-up

This option determines whether or not Picard will automatically check for program updates during startup. In any case, you can have Picard check for program updates at any time using *"Help → Check for update"*.

Days between checks

This option allows you to limit the automatic update checking by selecting the interval, in days, between checks. Set this to 1 if you want to check daily, 7 for weekly checks, and so on. Note that this only applies if the "Check for updates during start-up" option is enabled.

Updates to check

This option allows you to select which levels of update to check. Your options are:

- Stable releases only

- Stable and Beta releases

- Stable, Beta and Dev releases

For example, if you subscribe to "Stable releases only" you will not be notified if a new Beta or Dev release is issued.

Note: The update checking related settings and *"Help → Check for update. . . "* command may not be available when Picard is distributed as a package. In that case, the user should check with the maintainer of the package to determine when an update is available.

4.3.2 Metadata Options

Translate artist names to this locale where possible

> When checked, Picard will see whether an artist has an alias for the selected locale. If it does,
> Picard will use that alias instead of the artist name when tagging. When "English" is the selected
> locale, the artist sort name (which is, by Style Guideline, stored in Latin script) is used as a
> fallback if there is no English alias.

Use standardized artist names

> Check to only use standard Artist names, rather than Artist Credits which may differ slightly
> across tracks and releases.

> **Note:** If the "Translate artist names" option above is also checked, it will override this option
> if a suitable alias is found.

Use standardized instrument and vocal credits

> Check to only use standard names for instruments and vocals in performer relationships.
> Uncheck to use the instruments and vocals as credited in the relationship.

Convert Unicode punctuation characters to ASCII

Converts Unicode punctuation characters in MusicBrainz data to ASCII for consistent use of punctuation in tags. For example, right single quotation marks are converted to ASCII apostrophes ('), and horizontal ellipses are converted to three full stops (. . .).

Use release relationships

Check to retrieve and write release-level relationships (e.g.: URLs, composer, lyricist, performer, conductor, or DJ mixer) to your files. You must have this enabled to use Picard to retrieve cover art.

Use track relationships

Check to write track-level relationships (e.g.: composer, lyricist, performer, or remixer) to your files.

Various artists

Choose how you want the "Various Artists" artist spelled.

Non-album tracks

Choose how you want "non-album tracks" to be grouped.

Preferred Releases
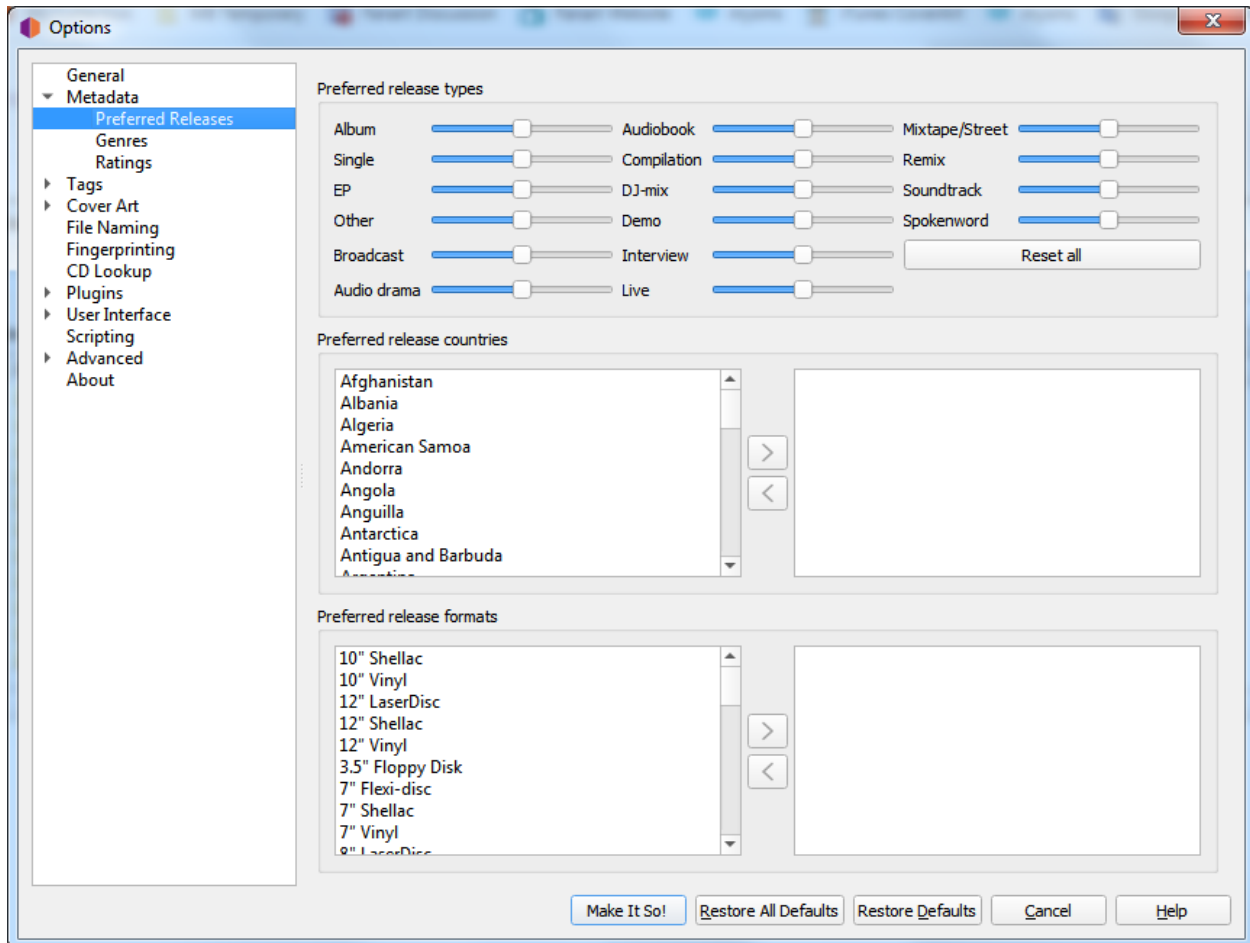
Preferred release types

Adjust the sliders for various release types to tweak how likely Picard is to match a file or cluster to releases of various types. For example, you can use this to decrease the likelihood of Picard matching a file or album to a Compilation or Live version.

Preferred release countries

Add one or more countries into the list to make Picard prefer matching clusters or files to releases from the chosen countries. This list is also used to prioritize files in the "Other Releases" context menu.

Preferred release formats

Add one or more formats into the list to make Picard prefer matching clusters or files to releases of the specified format. This list is also used to prioritize files in the "Other Releases" context menu.

Genres

Use genres from MusicBrainz

Use genres provided by MusicBrainz and save them to the genre tag.

Fall back on album's artists genres if no genres are found for the release or release group

If there is no genre set for the release or release group on MusicBrainz, use the genre of the album artist instead.

Only use my genres

When enabled, Picard will only write genres you personally have submitted to MusicBrainz. You'll need to set your username and password to use this feature.

Use folksonomy tags as genres

Check to use all folksonomy tags to set the genre. Otherwise only the tags considered by MusicBrainz to be proper genres will be used.

Minimal genre usage

Choose how popular the genre must be before it is written by Picard. Default: 90%. Lowering the value here will lead to more, but possibly less relevant, genres in your files.

Maximum number of genres

Choose how many genres to use. Default: 5. If you only want a single genre, set this to 1.

Join multiple genres with

Select which character should be used to separate multiple genres.

Genres or folksonomy tags to include or exclude

One expression per line, case-insensitive. You can use the "Playground" text field to enter some genres and test the rules you have setup. Genres that will be excluded will be marked red, included genres will be marked green.

- **Comments**: Lines not starting with '-' or '+' are ignored. (e.g.: `#comment`, `!comment` or `comment`)

- **Strict filtering**: Exclude exact word by prefixing it with '-' (e.g.: `-word`). Include exact word, even if another rule would exclude it, by prefixing it with '+' (e.g.: `+word`).

- **Wildcard filtering**: Exclude all genres ending with "word" (e.g.: `-*word`). Include all genres starting with "word" (e.g.: `+word*`). Exclude all genres starting with 'w' and ending with "rd" (e.g.: `-w*rd`).

- **Regular expressions filtering (Python "re" syntax)**: Exclude genres starting with 'w' followed by any character, then 'r' followed by at least one 'd' (e.g.: `-/^w.rd+/`).

Playground for genres or folksonomy tags filters:

This area allows you to enter genre tags, one per line, to test your filters. If a tag is marked in red, it will be filtered out. A tag marked green will be allowed.

Note: This list of test tags will be cleared when you exit the configuration section.

Ratings

Enable track ratings

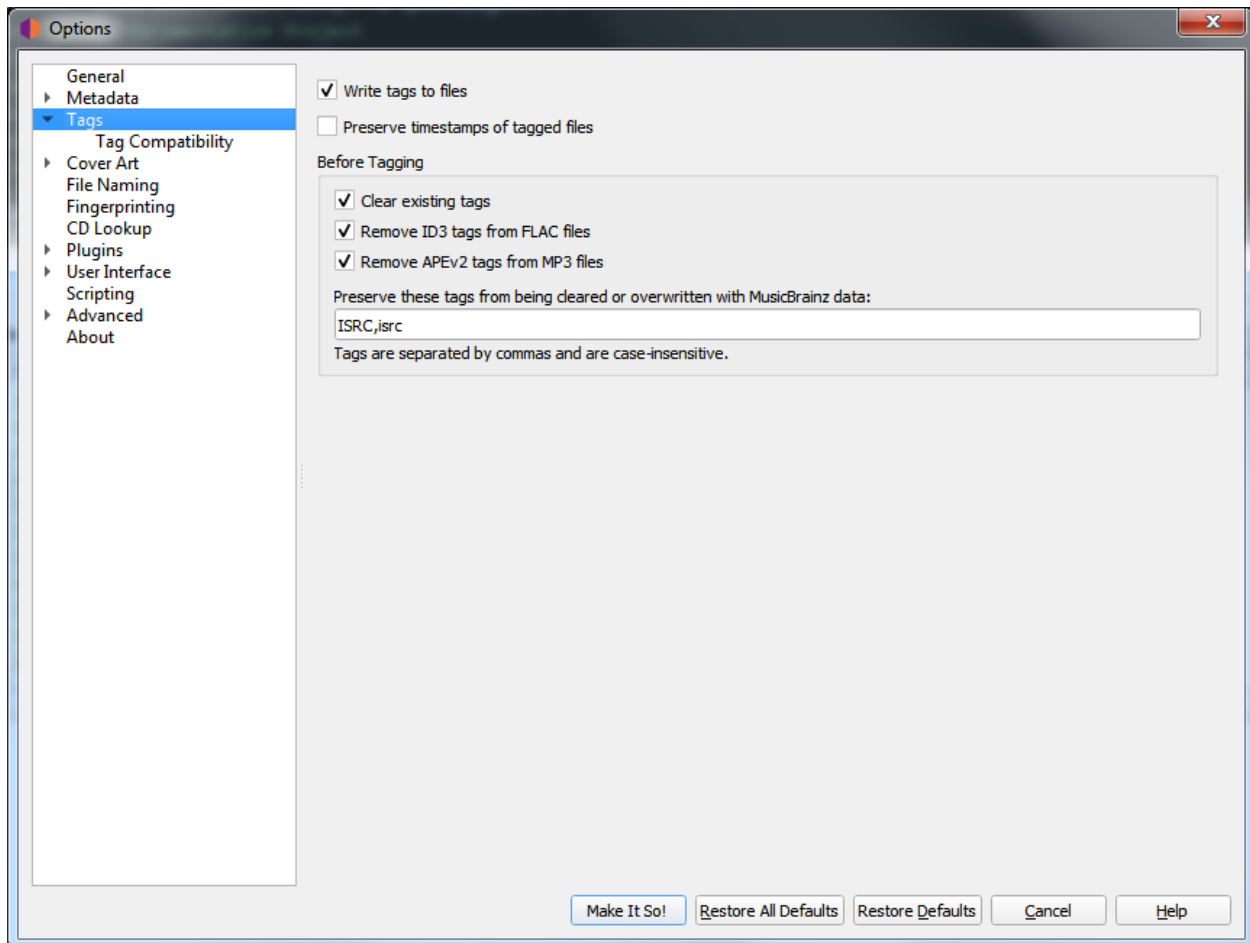   Check to write track ratings to your files.

E-mail

    The email address used when submitting ratings to MusicBrainz. This identifies the user that provided the rating.

Submit ratings to MusicBrainz

    Check to submit ratings to MusicBrainz. The tracks will be rated with your account.

4.3.3 Tag Options

Write tags to files

> Uncheck to disable Picard from writing metadata to your files. Picard may still move or rename your files according to your settings.

Preserve timestamps of tagged files

> If checked, Picard will not update the "Last Modified" date and time of your music files when it writes new tags to them.

Before Tagging

Clear existing tags

> Checking this will remove all existing metadata and leave your files with only MusicBrainz metadata. Information you may have added through another media player such as "genre", "comments" or "ratings" will be removed.

Remove ID3 tags from FLAC files

Check to remove ID3 tags from FLAC files – Vorbis Comments are recommended for FLAC files. Picard will write Vorbis Comments to FLAC files regardless of this setting.

Remove APEv2 tags from MP3 files

Check to remove APEv2 tags from MP3 files – ID3 is recommended for MP3 files. Picard will write ID3 tags to MP3 files regardless of this setting.

Preserve these tags from being cleared or overwritten with MusicBrainz data

This is an advanced option: If you have tags which you need to preserve, enter their names here to stop Picard from overwriting them.

Tag Compatibility
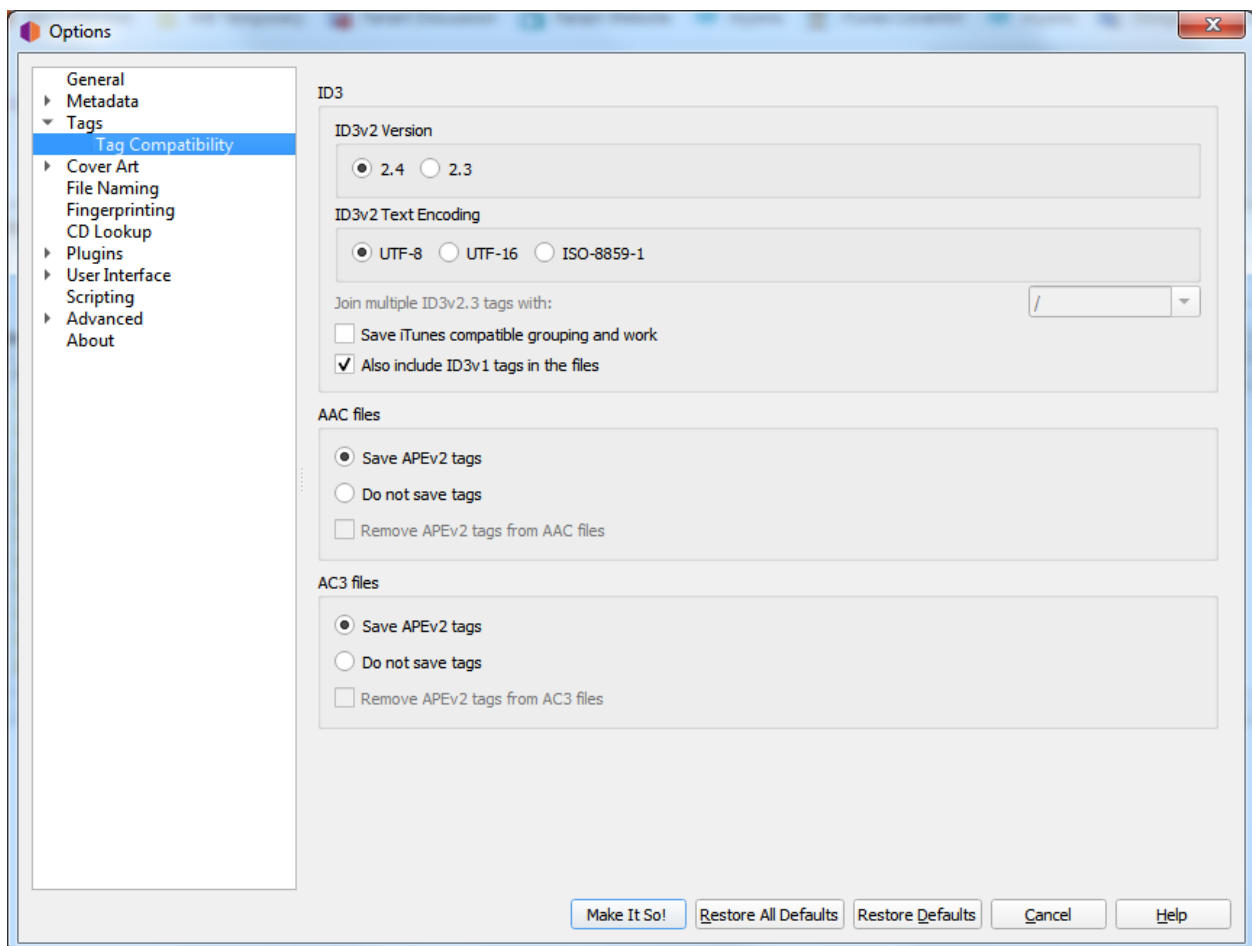
ID3v2 version

Although ID3v2.4 is the latest version, its support in music players is still lacking. While software such as foobar2000[42] and MediaMonkey[43] have no problem using version 2.4 tags, you will not be able to read the tags in Windows Explorer or Windows Media Player (in any

---

[42] https://www.foobar2000.org/
[43] https://www.mediamonkey.com/

Windows or WMP version). Apple iTunes is also still based in ID3v2.3, and support for ID3v2.4 in other media players (such as smartphones) is variable. Other than native support for multi-valued tags in v2.4, the Picard Tag Mapping[44] will show you what you lose when choosing v2.3 instead of v2.4.

ID3v2 text encoding

The default for version 2.4 is UTF-8, the default for version 2.3 is UTF-16. Use ISO-8859-1 only if you face compatibility issues with your player.

Join ID3v23 tags with

As mentioned above, ID3v2.3 does not support multi-value tags, and so Picard flattens these to strings before saving them to ID3v2.3 tags. This setting defines the string used to separate the values when flattened. Use '; ' for the greatest compatibility (rather than '/' since tags more often contain a '/' than a ';') and for the best visual compatibility in Picard between ID3v2.3 and other tagging formats.

Save iTunes compatible grouping and work

Save the tags grouping and work so that they are compatible with current iTunes versions. Without this option grouping will be displayed in iTunes as "work name" and work will not be available. See the Picard Tag Mapping page[45] for details.

Note: For other players supporting grouping and work you might need to disable this option. MusicBee[46] is one example of this.

Also include ID3v1 tags in the files

This is not recommended at all. ID3v1.1 tags are obsolete and may not work with non-latin scripts.

AAC / AC3 files

Picard can save APEv2 tags to pure AAC or AC3 files, which by default do not support tagging. APEv2 tags in AAC or AC3 are supported by some players (e.g.: foobar2000 or MusicBee), but players not supporting AAC or AC3 files with APEv2 tags can have issues loading and playing those files. Most often they display a wrong duration, causing issues on track change. To deal with this you can choose whether to save tags to those files:

- **Save APEv2 tags**: Picard will save APEv2 tags to the files.

- **Do not save tags**: Picard will not save any tags to the files, but you can still use Picard to rename them. By default existing APEv2 tags will be kept in the file.

- **Remove APEv2 tags**: If you have "Do not save tags" enabled checking this option will cause Picard to remove existing APEv2 tags from the file on saving.

Regardless of how you have configured saving tags Picard will always read existing APEv2 tags in AAC or AC3 files.
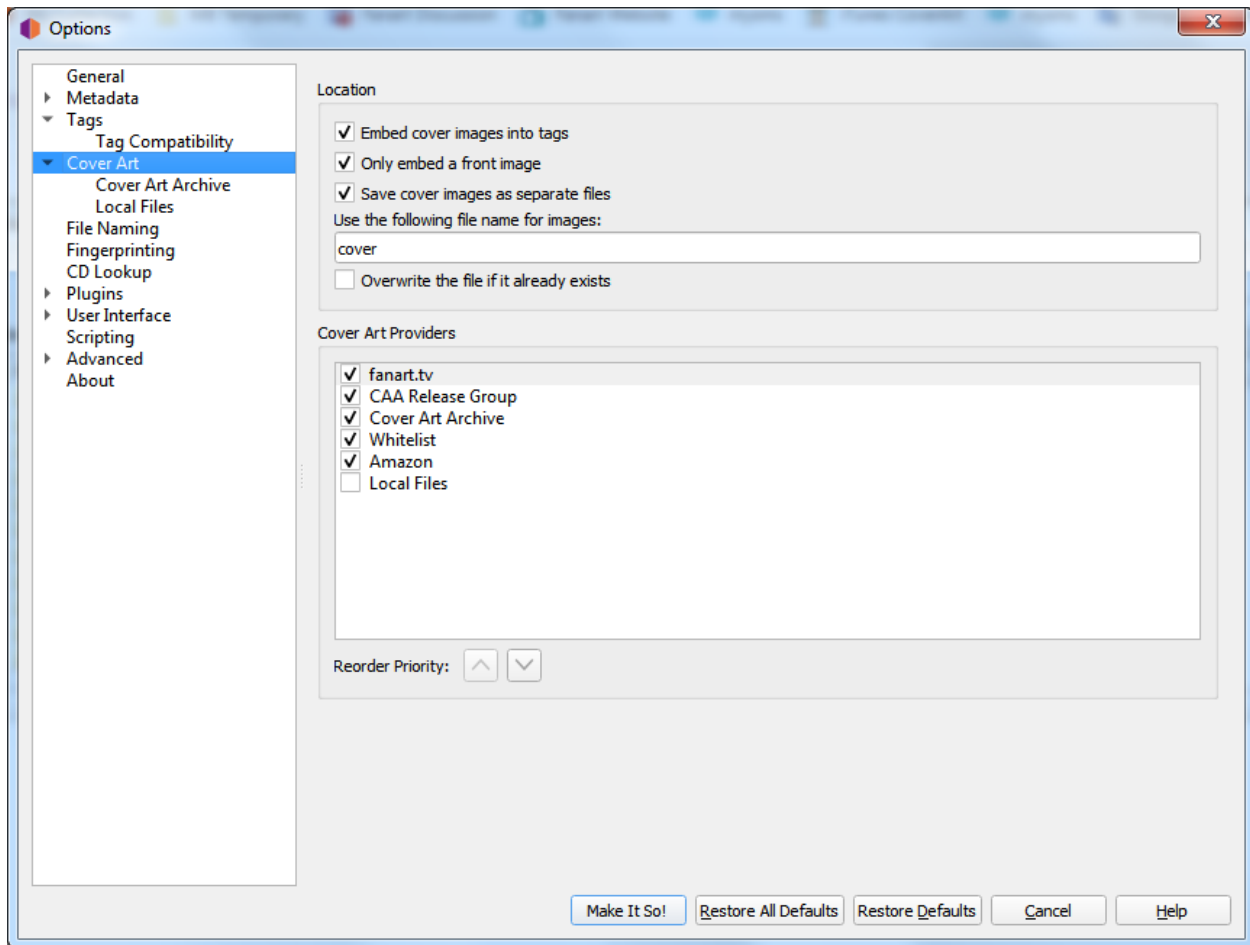
---

[44] https://picard.musicbrainz.org/docs/mappings/
[45] https://picard.musicbrainz.org/docs/mappings/
[46] https://getmusicbee.com/

4.3.4 Cover Art Options

Note: You must enable *"Options → Metadata → Use release relationships"* for Picard to be able to download cover art from MusicBrainz cover art relationships.

Location

Embed cover images into tags

Enables images to be embedded directly into your music files. While this will use more storage space than storing it as a separate image file in the same directory, some music players will only display embedded images and don't find the separate files.

Only embed a front image

Embeds only a front image into your music files. Many music players will only display a single embedded image, so embedding additional images may not add any functionality.

Save cover images as separate files

In the file name mask you can use any variable or function from *Picard Tags* and *Picard Scripting Functions*. The mask should not contain a file extension; this is added automatically based on the actual image type. The default value is "cover". If you change this to "folder", Windows will use it to preview the containing directory.

In addition to scripting variables already available for a track you can use the following cover art specific variables:

- `coverart_maintype`: The primary type (e.g.: front, medium, booklet). For front images this will always be "front".
- `coverart_types`: Full list of all types assigned to this image.
- `coverart_comment`: The cover art comment.

Overwrite the file if it already exists

Check this to replace existing files. This is especially recommended if trying to write "folder" previews for Windows.

Cover Art Providers

Picard can download Cover Art from a number of sources, and you can choose which sources you want Picard to use. You can activate more than one provider and choose the order in which the providers are queried. Picard will try the providers from top to bottom until an image is returned.

Cover Art Archive

The Cover Art Archive (CAA) is the MusicBrainz archive of cover art in cooperation with the Internet Archive[47]. The Cover Art Archive is the most comprehensive database of cover art (e.g.: front covers, back covers, booklets, CDs).

CAA Release Group

This provider uses the Cover Art Archive cover image assigned to the release group. This is usually the image that best describes the release group as a whole or the image with the best visual quality, but is not necessarily the exact cover of the release you are tagging. This provider is a good choice if you care more about visual quality then having an exact representation of your release. It is also a good fallback for the Cover Art Archive provider.

Sites on the whitelist

This will use images provided from whitelisted sites. See Cover art whitelist[48] in the Style Guide for more information.

Note: CD Baby and other whitelist sites are no longer being used by MusicBrainz for new Cover Art.


Local Files

---

[47] https://archive.org
[48] https://wiki.musicbrainz.org/History:Style/Relationships/URLs/Cover_art_whitelist

Load cover art from local files. The file names to load can be configured in the *Local Files* provider options.

In addition to the built-in cover art providers described above, additional cover art providers can be installed as plugins[49].

- **Amazon**: Amazon often has cover art when other sites don't, however while this art is almost always for the correct Artist and Album, it may not be the absolute correct cover art for the specific Release with which you have tagged your music. *Note: The Amazon cover art provider was built-in in Picard 2.1.3 and earlier versions. For later versions it needs to be installed as a separate plugin.*

- **fanart.tv**: Uses cover art from fanart.tv[50], which focuses on cover art with high visual quality. This provider provides cover art representative for the release group and not the individual release.

- **TheAudioDB**: Uses cover art from TheAudioDB[51], which focuses on cover art with high visual quality. This provider provides cover art representative for the release group and not the individual release.

Cover Art Archive

---

[49] https://picard.musicbrainz.org/plugins/
[50] https://fanart.tv/
[51] https://www.theaudiodb.com/
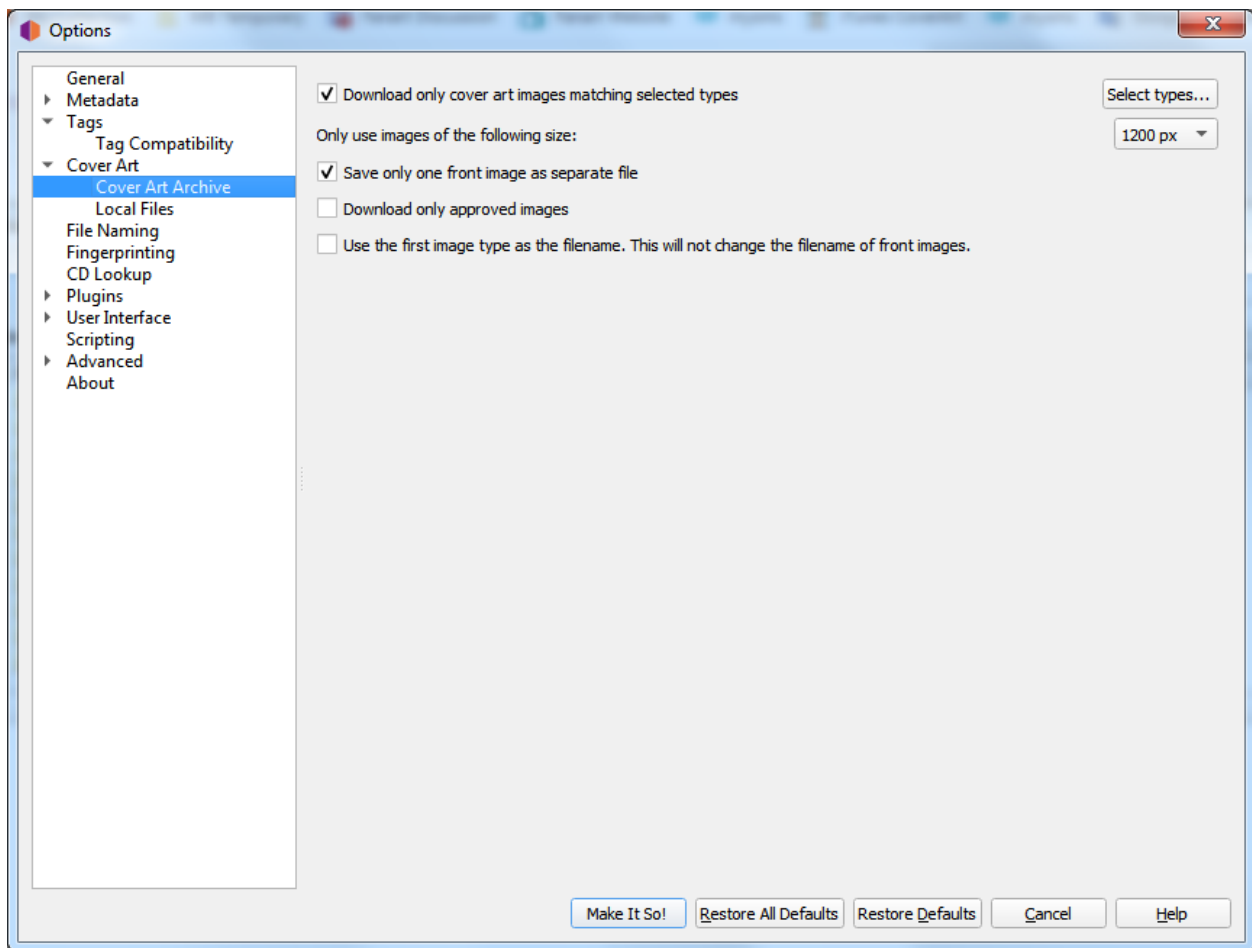
In this section you can decide which types of cover art you would like to download from the Cover Art Archive, and what quality (size) you want to download. Obviously, the better the quality, the larger the size of the files.

Download Types

When selecting the cover art image types, you can select the types to both include and exclude from the download list. CAA images with an image type found in the "Include" list will be downloaded and used unless they also have an image type found in the "Exclude" list. Images with types found in the "Exclude" list will never be used. Image types not appearing in either the "Include" or "Exclude" lists will not be considered when determining whether or not to download and use a CAA image.

Most music players will display only one piece of cover art for the album, and most people select Front (cover) for that.

Image Size

This identifies what size of image to download from the CAA. The options are 250px, 500px, 1200px amd full size. The fixed sizes are generated automatically from the full size image, provided that it is greater than or equal to the fixed size being generated. The generated images are square and padded as required if the original image is not square.

Save only one front image

This tells Picard to only save the first "front" image to a separate file with the release. If left unchecked, all "front" images will be saved as separate files.

Download only approved images

When checked, Picard will only download images that have been approved (i.e.: the edit to add the image has been accepted and applied). To allow using images from pending edits, leave this option unchecked.

Use the first image type as the filename

When checked, Picard will use the type of the first image retrieved as the filename when saving all images. If left unchecked, each file will be named according to its image type.


Note: This will not change the name used for "front" images that has been specified in the *Save cover images* section of the general *"Cover Art Options"*.

Since Picard 1.3, you can also decide whether or not to use the image from the release group (if any) if no front image is found for the release. In this case, the cover may not match the exact release you are tagging (eg.: a 1979 vinyl front cover may be used in place of the Deluxe 2010 CD reissue).

Local Files
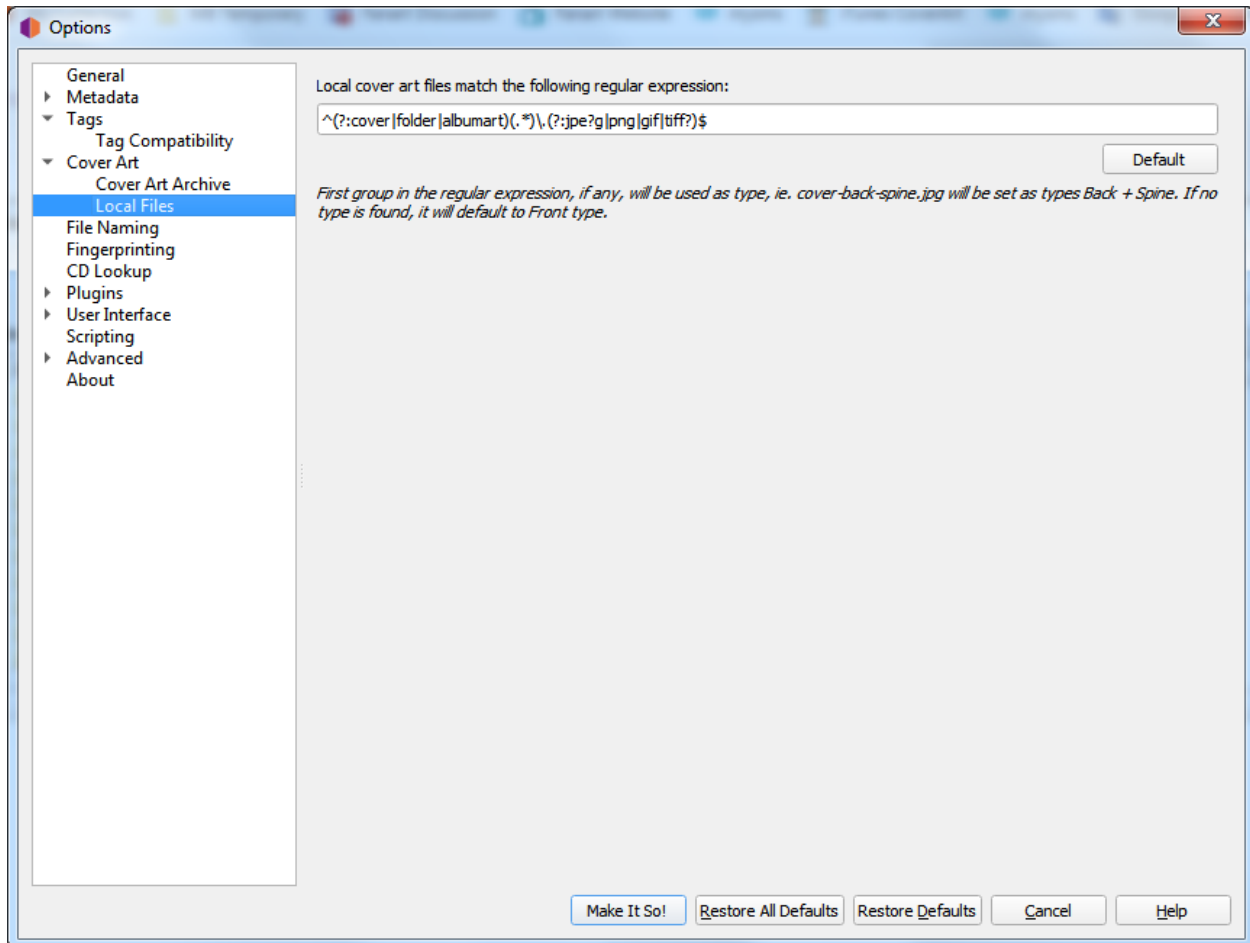
In this section you can configure the file names to be used by the Local Files cover art provider. The file names are defined using a regular expression. The default is to load files with the name "cover", "folder" or "albumart" and the file extension "jpg", "png", "gif" or "tiff" (e.g.: "folder.jpg" or "cover.png").

4.3.5 File Naming Options

These options determine how Picard handles files when they are saved with updated metadata.

Move files when saving

If selected, this option tells Picard to move your audio files to a new directory when it saves them. One use for this is to keep your work organized: all untagged files are under "Directory A", and when Picard tags them it moves them to "Directory B". When "Directory A" is empty, your tagging work is done.

If this option is left unchecked, then Picard will leave the files in the same directory when they are saved.

Note that the "Rename Files" and "Move Files" options are independent of one another. "Rename Files" refers to Picard changing file names, typically based on artist and track names. "Move Files" refers to Picard moving files to new directories, based on a specified parent directory and subdirectories, typically based on album artist name and release title. However, they both use the same "file naming string". "Move files" uses the portion up until the last '/'. "Rename files" uses the portion after the last '/'.

Destination directory

This specifies the destination parent diretory to which files are moved when they are saved, if

the "Move files when saving" option is selected. If you use the directory "." the files will be moved relative to their current location. If they are already in some sort of directory structure, this will probably not do what you want!

Move additional files

Enter patterns that match any other files you want Picard to move when saving music files (e.g.: "Folder.jpg", "*.png", "*.cue", "*.log"). Patterns are separated by spaces. When these additional files are moved they will end up in the release directory with your music files. In a pattern, the '*' character matches zero or more characters. Other text, like ".jpg", matches those exact characters. Thus "*.jpg" matches "cover.jpg", "liner.jpg", "a.jpg", and ".jpg", but not "nomatch.jpg2".

Delete empty directories

When selected, Picard will remove directories that have become empty once a move is completed. Leave this unchecked if you want Picard to leave the source directory structure unchanged. Checking this box may be convenient if you are using the "move files" option to organize your work. An empty directory has no more work for you to do, and deleting the directory makes that clear.

Rename files when saving

Select this option to let Picard change the file and directory names of your files when it saves them, in order to make the file and directory names consistent with the new metadata.

Replace non-ASCII characters

Select this option to replace non-ASCII characters with their ASCII equivalent (e.g.: 'á', 'ä' and 'ǎ' with 'a'; 'é', 'ě' and 'ë' with 'e'; 'æ' with "ae"). More information regarding ASCII characters can be found on Wikipedia[52].

Windows compatibility

This option tells Picard to replace all Windows-incompatible characters with an underscore. This is enabled by default on Windows systems, with no option to disable.

Name files like this

An edit box that contains a formatting string that tells Picard what the new name of the file and its containing directories should be in terms of various metadata values. The formatting string is in *Picard's scripting language* where dark blue text starting with a '$' is a *function name* and names in light blue within '%' signs are Picard's *tag and variable names*, and is generally referred to as a "file naming script". Note that the use of a '/' in the formatting string separates the output directory from the file name. The formatting string is allowed to contain any number of '/' characters. Everything before the last '/' is the directory location, and everything after the last '/' becomes the file's name.

There is only one file naming script defined in a user's settings, although it can vary from a simple one-line script such as `%album%/%title%` to a very complex script using different file naming formats based on different criteria. In all cases, the files will be saved using the text output by the script.
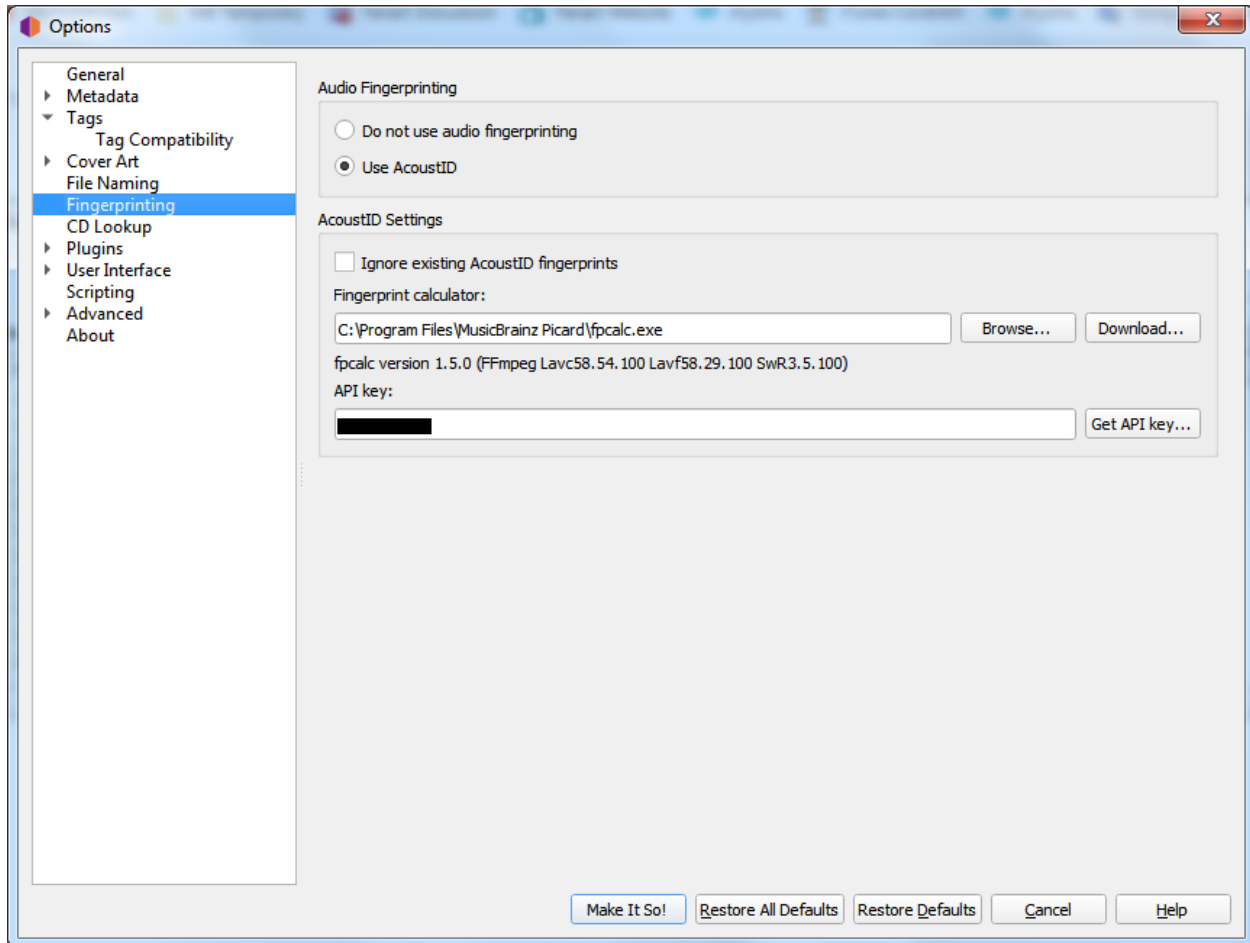
---

[52] https://en.wikipedia.org/wiki/ASCII

Scripts are often discussed in the MetaBrainz Community Forum[53], and there is a thread specific to file naming and script snippets[54].

Note: Any new tags set or tags modified by the file naming script will not be written to the output files' metadata.

4.3.6 Fingerprinting Options

If you select a file or cluster in the left-hand side of the Picard screen and select *"Tools → Scan"*, Picard will invoke a program to scan the files and produce a fingerprint for each that can then be used to look up the file on MusicBrainz.

MusicBrainz currently supports only AcoustID[55] (an Open Source acoustic fingerprinting[56] system created by Lukáš Lalinský[57]) but has previously supported TRM and MusicID PUID.

---

[53] https://community.metabrainz.org/
[54] https://community.metabrainz.org/t/repository-for-neat-file-name-string-patterns-and-tagger-script-snippets/2786/
[55] https://musicbrainz.org/doc/AcoustID
[56] https://musicbrainz.org/doc/Fingerprinting
[57] https://oxygene.sk/

Audio Fingerprinting

> This allows you to select whether or not to enable acoustic fingerprinting within Picard. If acoustic fingerprinting is diabled then all remaining options in this tab will be locked and ignored.

Ignore existing AcoustID fingerprints

> When checked, any existing AcoustID fingerprint information will not be used, and the files will be rescanned.

Fingerprint calculator

> This identifies the external program on your system that will be used to calculate the AcoustID fingerprints. By default, Picard uses the Chromaprint[58] (fpcalc) utility which is included with the Picard installation.

API key

> The key used to access the AcoustID API to lookup and submit AcoustID fingerprints. There is no cost to obtain an API key.

4.3.7 CD Lookup Options

This section allows you to select which CD ROM device to use by default for looking up a CD.

On Windows and Linux systems, you can override this setting by clicking on *"Tools → Lookup CD. . . "* and selecting the desired device from the list of available devices.

---

[58] https://acoustid.org/chromaprint

Windows

On Windows, Picard has a pulldown menu listing the various CD drives it has found. Pull down the menu and select the drive you want to use by default.

You can override this setting by clicking on *"Tools → Lookup CD. . . "* and selecting the desired device from the list of available devices.

macOS

In macOS, this option is currently a text field. The device is usually `/dev/rdisk1`.

If that doesn't work, one way is to simply keep increasing the number (e.g. `/dev/rdisk2`) until it does work. A less trial and error method is to open "Terminal" and type `mount`. The output should include a line such as:

```
/dev/disk2 on /Volumes/Audio CD (local, nodev, nosuid, read-only)
```

You need to replace `/dev/disk` with `/dev/rdisk`, so if, for example, it says `/dev/disk2`, you should enter `/dev/rdisk2` in Picard's preferences.

Linux

In Linux, Picard has a pulldown menu like in Windows. If you're using an older version of Picard with a text field, you should enter the device name (typically `/dev/cdrom`).

You can override this setting by clicking on *"Tools → Lookup CD..."* and selecting the desired device from the list of available devices.

Other platforms

On other platforms, the CD Lookup option is a text field and you should enter the path to the CD drive here.

4.3.8 Plugins Options
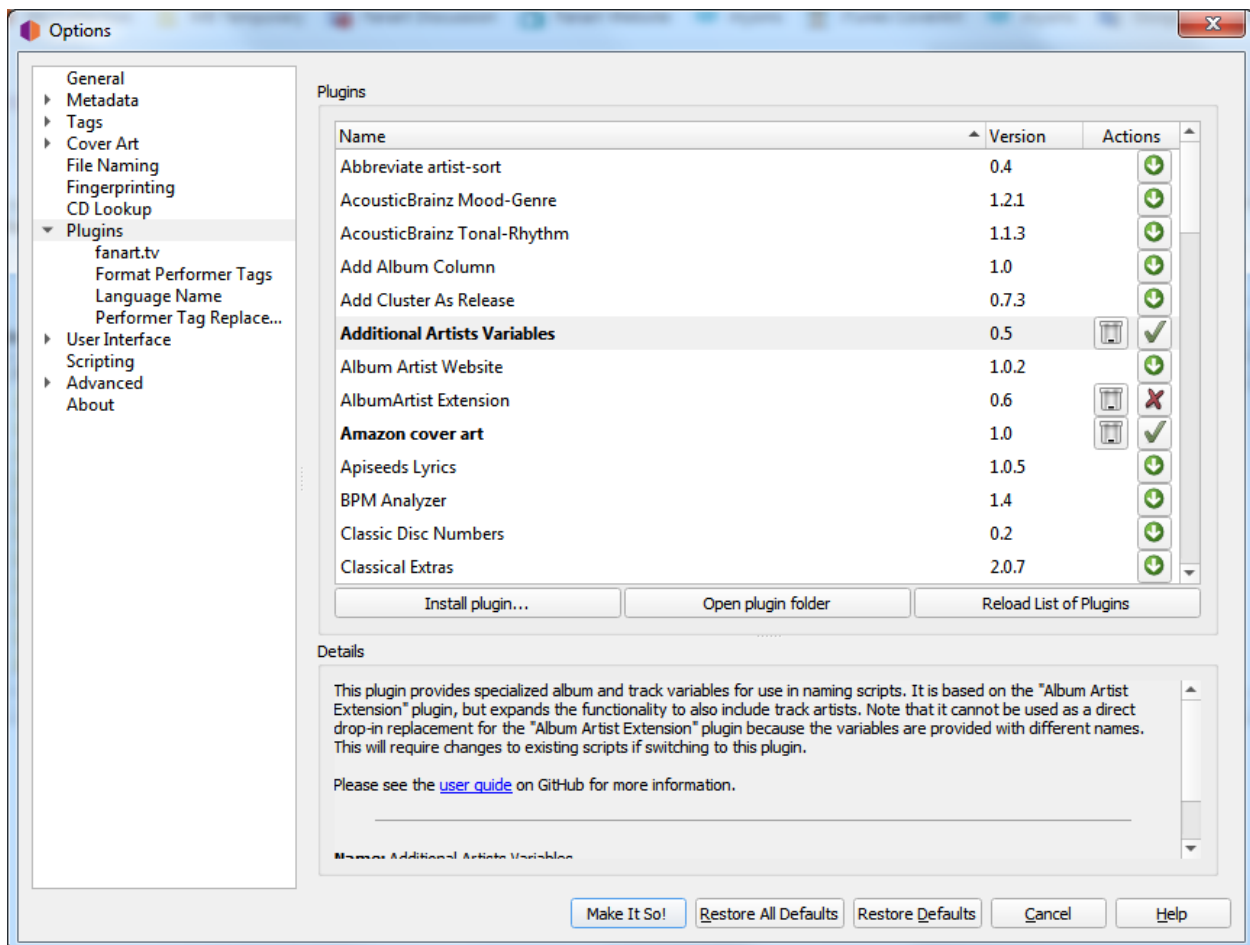
This section allows you to manage the plugins used by Picard. You can install new plugins or enable, disable or uninstall plugins that are currently installed. Picard provides a list of plugins that have been submitted to the project. A list of the standard plugins is available on the plugins page[59] on the Picard website.

---

[59] https://picard.musicbrainz.org/plugins/

There are also a number of plugins available by third-party developers. Often these are discussed on the Community Discussion Forum[60] so if you're looking for a particular enhancement or functionality, a search there might be useful. In addition, one of the MusicBrainz editors, Colby Dray[61] maintains an unofficial list of available plugins on a wiki page[62]. Instructions regarding installation of third-party plugins are included in the *"Installing Third-Party Plugins"* section below.

Plugins List

The screen displays a list of the standard plugins and any others that have been installed. Each plugin is displayed on a separate line showing the version number and one or more status / action icons. The icons are:

This icon indicates that the plugin is not installed. Clicking the icon will download and install the plugin.

This icon indicates that a newer version of the plugin is available. Clicking the icon will download and install the updated version.

This icon indicates that the plugin is installed and currently enabled. Clicking the icon will disable the plugin, but it will still be installed.

This icon indicates that the plugin is installed but currently disabled. Clicking the icon will enable the plugin.

This icon indicates that the plugin is currently installed. Clicking the icon will uninstall the plugin.
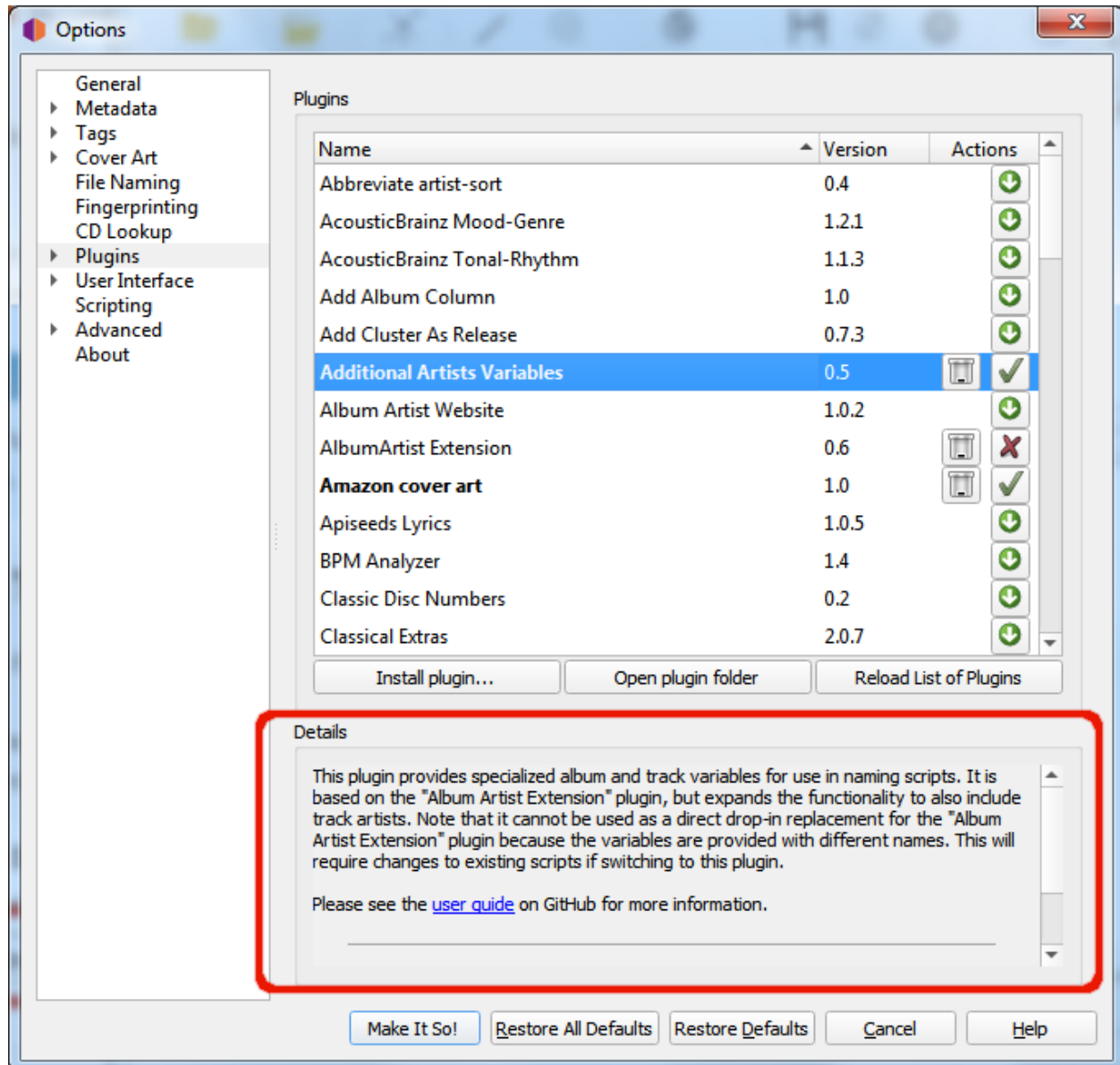
When a plugin in the list is selected (i.e.: highlighted), a brief description of the plugin will be shown in the "Details" section below the list.

---

[60] https://community.metabrainz.org/
[61] https://wiki.musicbrainz.org/User:Colbydray
[62] https://wiki.musicbrainz.org/User:Colbydray/PicardPlugins

Note: Some plugins have their own option page which will typically appear under the "Plugins" section of the Options.

Installing Third-Party Plugins

You can install a third-party plugin that does not appear in the plugins list, using the following steps:

1. Download the plugin and save it to a local drive.
2. Select the "Install plugin…" action, located just below the list of plugins.

3. Navigate to the file you downloaded in Step 1 and select it. The file will be copied to the plugin folder, and will appear in the list of plugins.
4. Enable the plugin if desired, and select the "Make It So!" action button at the bottom of the window.

4.3.9 User Interface Options
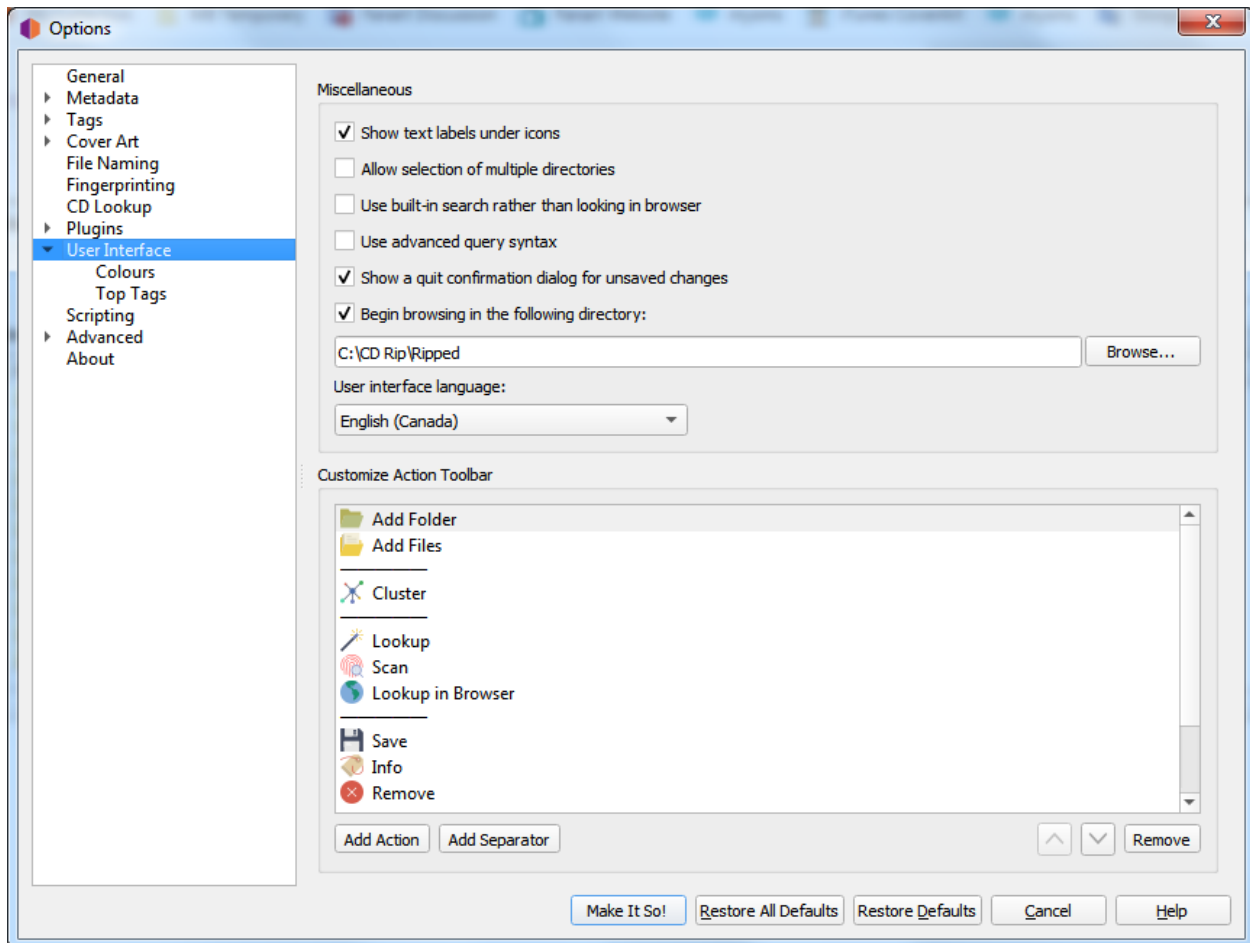
Show text labels under icon

> If this option is disabled, the text labels under the icons in the toolbar will not be displayed, causing the toolbar to appear a little smaller.

Allow selection of multiple directories

> Enabling this option will bypass the native directory selector and use QT's file dialog. This may be desirable since the native directory selector generally doesn't allow you to select more than one directory. This applies to the *"File → Add folder"* dialog. The file browser always allows multiple directory selection.

Use advanced query syntax

> This will enable advanced query syntax parsing on your searches. This only applies to the search box at the top right of Picard, not the lookup buttons.

Show a quit confirmation dialog for unsaved changes

> When this is enabled, Picard will show a dialog when you try to quit the program with unsaved files loaded. This may help prevent accidentally losing tag changes you've made, but not yet saved.

Begin browsing in the following directory

> By default, Picard remembers the last directory used to load files. If you enable this option and provide a directory, Picard will always start in the directory provided.

User interface language

> By default, Picard will display in the language displayed by your operating system, however you can override this and select a different language if needed.

Customize action toolbar

> This allows you to to add, remove or rearrange the items displayed in the Action Toolbar.

Colors

This section allows you to customize the various colors used in the Picard user interface. To change a color, simply click on the color block currently displayed for the desired text condition to bring up a selection dialog, then pick your desired color. The colors can be changed for the following text conditions:

- **Errored entity**: files and other elements with errors on loading or saving

- **Pending entity**: files and other elements queued up for processing

- **Saved entity**: successfully saved files

- **Log view text (debug)**: debug messages in the Error/Debug Log

- **Log view text (error)**: error messages in the Error/Debug Log

- **Log view text (info)**: informational messages in the Error/Debug Log

- **Log view text (warning)**: warning messages in the Error/Debug Log

- **Tag added**: newly added tags in the metadata pane

- **Tag changed**: changed tags in the metadata pane

- **Tag removed**: removed tags in the metadata pane

Top Tags

The tags specified in this option setting will always be shown in the specified order at the top of the metadata pane (which shows the metadata of selected files or tracks). This allows you to have the most important tags always on top of the list. Tags not listed here will be shown in alphabetical order below the top tags.

4.3.10 Scripting Options
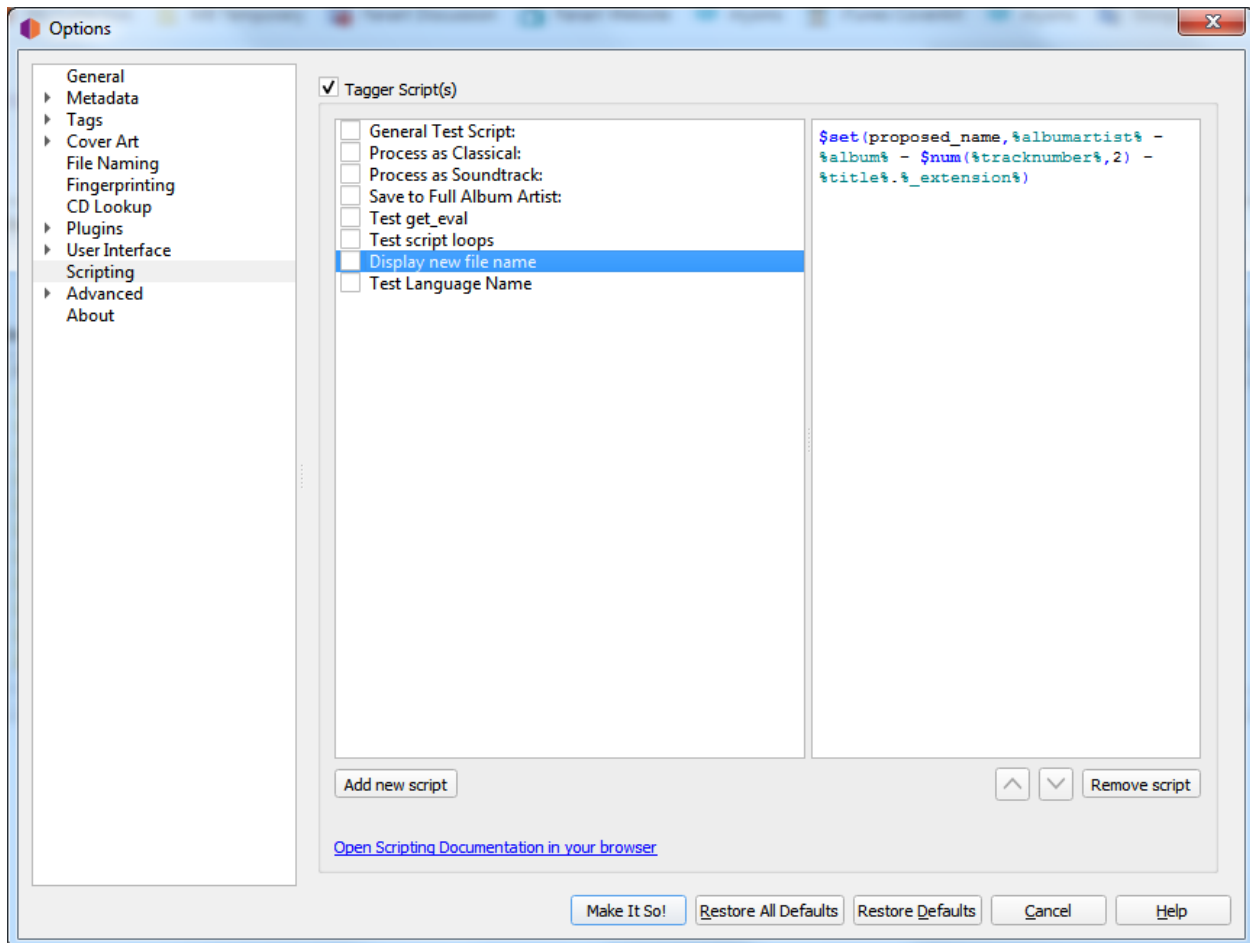
This section allows for the management of user-defined tagging scripts.

The "Tagger Script(s)" checkbox at the top of the page allows you to completely disable all tagging scripts. This can be useful when tracking down a problem with Picard's configuration.

Below the checkbox are two columns showing the list of scripts in the left-hand column, with the content of the selected script shown in the right-hand column. This section allows you to add, remove and reorder the scripts, enable or disable individual scripts, as well as edit the currently selected script.

For additional information about scripting please see the "*Scripts*" and "*Scripting*" sections, as well as "*Tags & Variables*".

4.3.11 Advanced Options
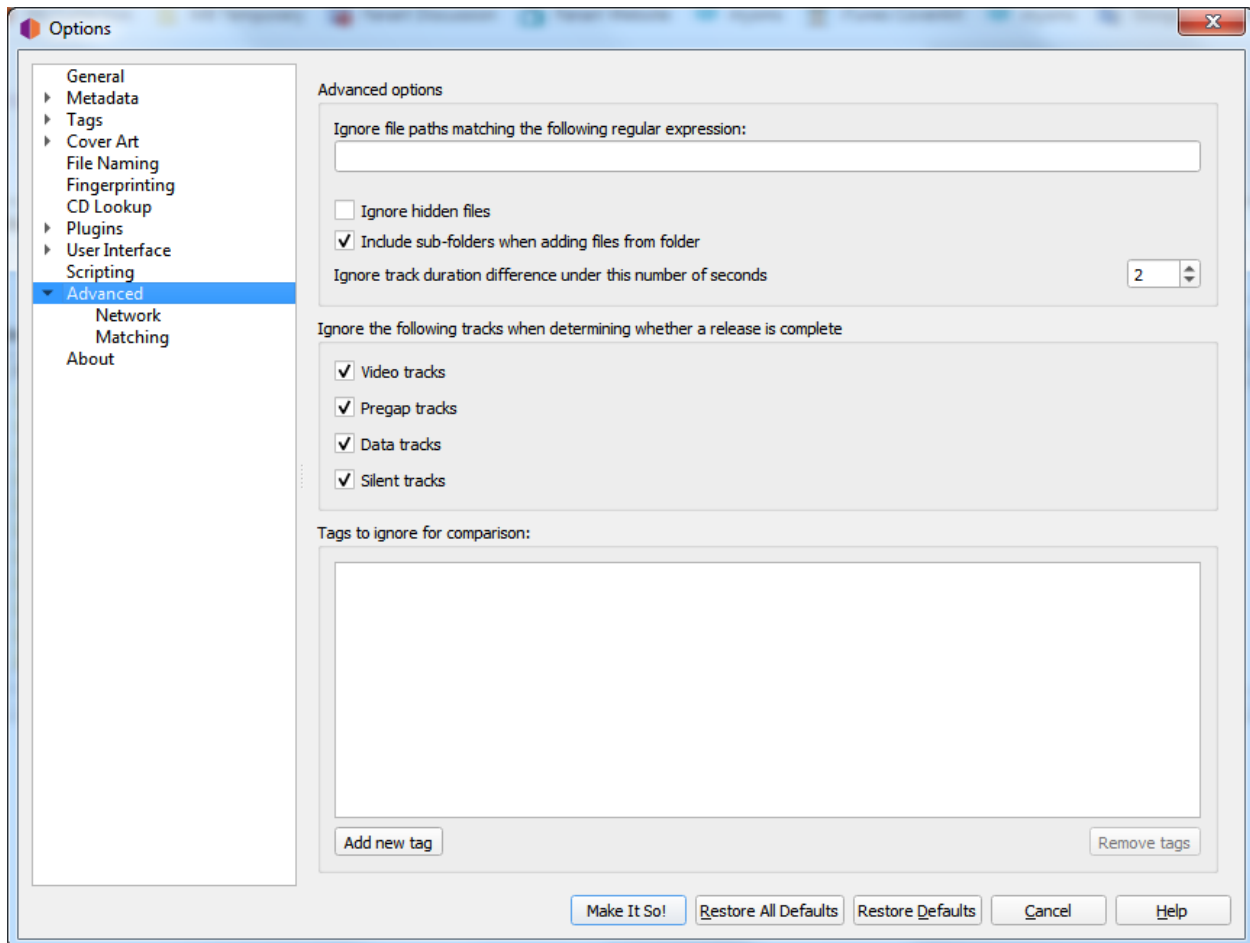
Ignore file paths matching the following regular expression

> You can specify patterns for files and directories that Picard should never load. For example, if
> you set this to the regular expression `\.bak$` any file ending in ".bak" will be ignored when
> loading files.

Ignore hidden files

> If this option is enabled then hidden files and directories will not be loaded. This also includes
> any file or subdirectory inside a hidden directory.

Include sub-folders when adding files from folders

> If this option is enabled Picard will load all audio files in the selected directory and all its
> subdirectories. If disabled only audio files in the selected directory will be loaded.

Ignore track duration difference under this number of seconds

> This specifies the number of seconds that a file can differ in length from the length in the
> MusicBrainz database and still be considered to be the same. The default value is 2 seconds.

Ignore the following tracks when determining whether a release is complete

Missing tracks of the selected type (i.e.: video, pregap, data or silence) will be ignored when determining whether a release is considered to be complete. For example, if "video" is selected then a release with a bonus video will be marked as complete if it has all the audio tracks matched with a file even if the video file is missing.

Tags to ignore for comparison

Tags in this list will not be considered when comparing the existing file metadata to the data retrieved from MusicBrainz. If the only difference between the file's metadata and the metadata retrieved from MusicBrainz is a tag listed in this ignore list then the file will be considered unmodified.

Network
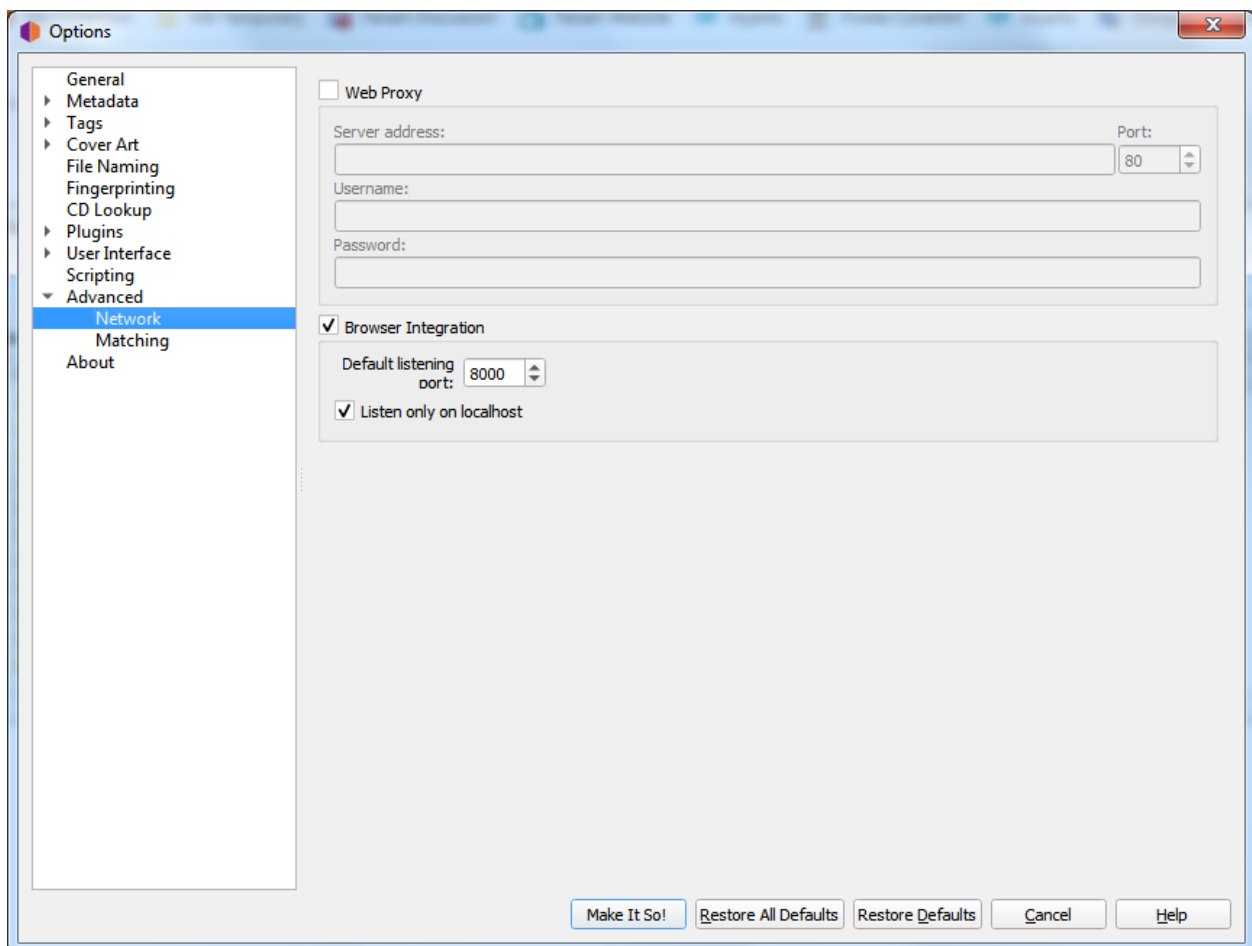
Web Proxy

If you need a proxy to make an outside network connection you may specify one here. The required settings are **Server Address**, **Port**, **Username** and **Password**.

Browser Integration

The browser integration allows you to load releases and recordings into Picard directly from the

---

MusicBrainz website. Once you have opened musicbrainz.org in your browser from Picard, the website will show the green tagger button ![TAGGER] next to releases and recordings. Clicking on this button will load the corresponding release or recording into Picard.

Default listening port

This identifies the default port Picard will listen on for the browser integration. If the port is not available Picard will try to increase the port number by one until it finds a free port.

Listen only on localhost

By default Picard will limit access to the browser integration port to your local machine. Deactivating this option will expose the port on your network, allowing you to request Picard to load a specific release or recording via the network. For example, this would be used for the Picard Barcode Scanner[63] Android app.
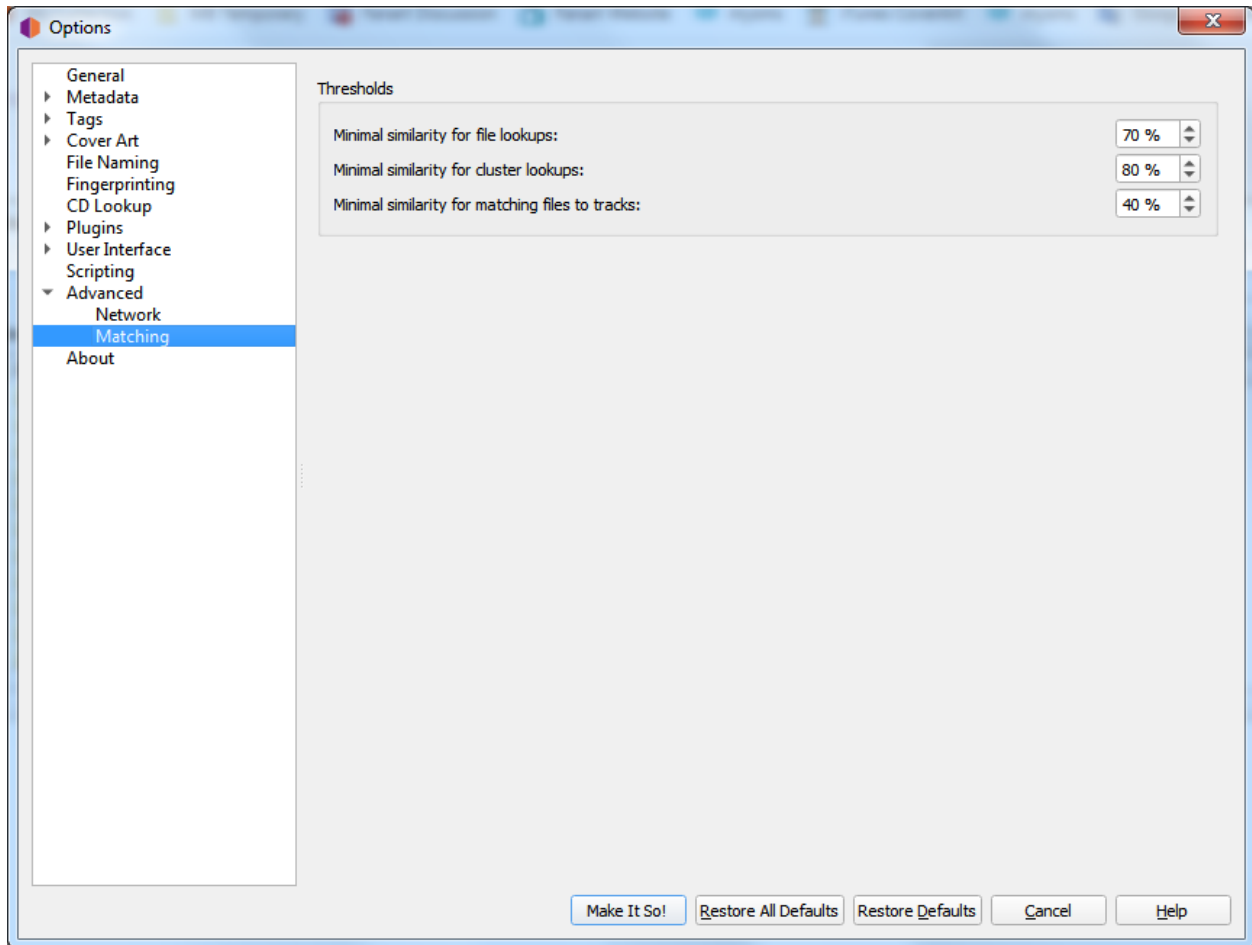
> **Warning:** Only expose the port externally when you actually need it and only on networks you trust. Exposing application ports via the network can open potential security holes on your system.

---

[63] https://play.google.com/store/apps/details?id=org.musicbrainz.picard.barcodescanner

Matching

It is recommended for most users to leave these settings at their default values. For advanced users, these allow you to tune the way Picard matches your files and clusters to MusicBrainz releases and tracks.

**Minimal similarity for file lookups**

> The higher the percentage value, the more similar an individual file's metadata must be to the metadata from MusicBrainz for it to be matched to a release on the right-hand side.

Minimal similarity for cluster lookups

> The higher the percentage value, the more similar a cluster of files from the left-hand pane must be to a MusicBrainz release for the entire cluster to be matched to a release on the right-hand side.

Minimal similarity for matching files to tracks

> The higher the percentage value, the more similar an individual file's metadata must be to the metadata from MusicBrainz for it to be matched to a release on the right-hand side.

If you have absolutely no metadata in your current files, and you are using "Scan" to match tracks, you may find that you need to lower the value of "Minimal similarity for matching files to tracks" in order for Picard

to match the files within a release. Otherwise you may find that Picard matches the track to a release but then is not sure which track is correct; and leaves it in an "unmatched files" group within that release.

As a general rule, lowering the percentages may increase the chance of finding a match at the risk of false positives and incorrect matches.

4.3.12 About
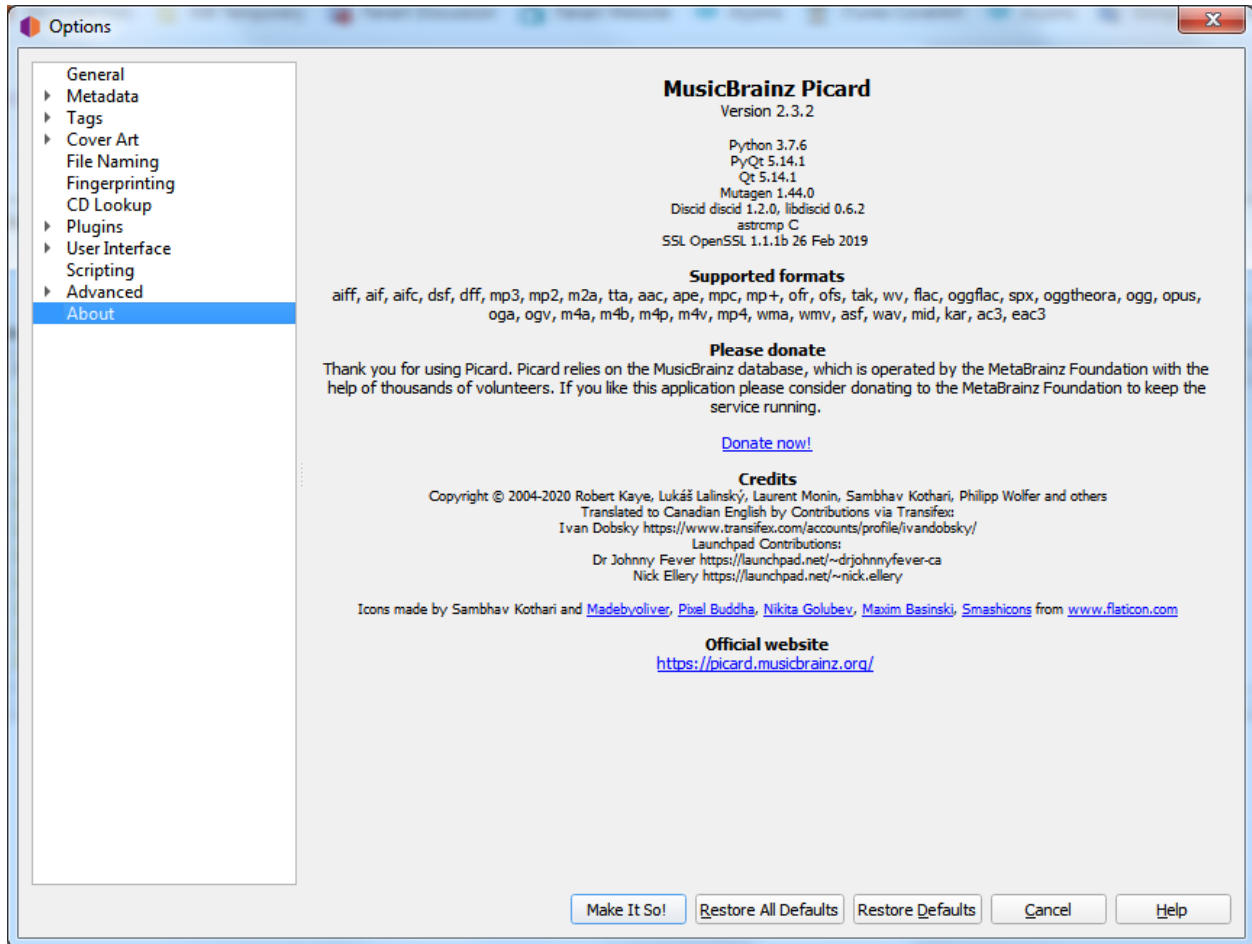
This section displays information about the currently installed version of Picard. This includes the version number of Picard and the libraries included, a list of the file types supported, key developer credits, and a link to the official website. This can also be displayed using *"Help → About. . . "*.

TAGS & VARIABLES

This describes both Tags which are saved inside your music files and can be read by your music player, and Picard variables which can be used in Picard scripts for tagging, file renaming, and in several other more minor settings.

All tags are also available as variables, but additional variables which start with an underscore '_' are not saved as Tags within your music files (e.g. `_my_tag_not_saved`).

Variables are used in scripts by wrapping the name between percent '%' characters (e.g. `%title%`).

Some variables can contain more than one value (e.g. `musicbrainz_artistid`), and if you want to use only one of the values then you will need to use special script functions to access or set them. To access all the multiple values at once, use the variable normally and Picard will combine them into a single string separated by a semicolon and space (e.g.: "Item 1; Item 2; Item 3").

If a tag description indicates a later version of Picard than the current official version on the downloads page, then the tag is beta functionality which will be available in the next official release. A description of how to gain access to these beta versions for testing can be found on the Picard downloads page[64] on the website.

5.1 Basic Tags

The following tags are populated from MusicBrainz data for most releases, without any special Picard settings.

All of these are also available as variables for use in Picard Scripts (for tagging, for file renaming and in several other more minor settings) by wrapping them between percent '%' symbols (e.g. `%title%`).

Note: Tags will not be created and will not be available as variables if there was no value retrieved for the tag from the MusicBrainz database.

Note: Some of these tags are only supported for certain file types or tag formats. Please see the Picard Tag Mapping[65] webpage for details.

---

[64] https://picard.musicbrainz.org/downloads/
[65] https://picard.musicbrainz.org/docs/mappings/

acoustid_fingerprint

AcoustID Fingerprint for the track.

acoustid_id

AcoustID associated with the track.

album

Title of the release.

albumartist

Artist(s) primarily credited on the release.

albumartistsort

Release Artist's Sort Name (e.g.: "Beatles, The").

albumsort

Release Title's Sort Name.

artist

Track Artist Name(s). (string)

artists

Track Artist Name(s). (multi-value) (*since Picard 1.3*)

artistsort

Track Artist Sort Name.

asin

Amazon Standard Identification Number - the number identifying the item on Amazon.

barcode

Release Barcode - the barcode assigned to the release.

bpm

Beats per minute of the track.

catalognumber

The number(s) assigned to the release by the label(s), which can often be found on the spine or near the barcode. There may be more than one, especially when multiple labels are involved.

comment

Disambiguation Comment - the comment entered to help distinguish one release from another (e.g.: Deluxe version with 2 bonus tracks).

compilation

(*since Picard 1.3, compatible with iTunes*) 1 for Various Artist albums, otherwise 0.

(*Picard 1.2 or previous*) 1 if multiple track artists (including featured artists), otherwise 0.

copyright

>Contain copyright message for the copyright holder of the original sound, begin with a year and a space character.

date

>Release Date (YYYY-MM-DD) - the date that the release was issued.

discid

>Disc ID is the code number which MusicBrainz uses to link a physical CD to a release listing. This is based on the table of contents (TOC) information read from the disc.

discnumber

>Number of the disc in this release that contains this track.

discsubtitle

>The Media Title given to a specific disc.

encodedby

>Encoded by (person or organization).

encodersettings

>Encoder Settings used.

isrc

>International Standard Recording Code - an international standard code for uniquely identifying sound recordings and music video recordings. See Wikipedia[66] for more information. (*since Picard 0.12*)

key

>Key of the music.

label

>Release Label Name(s).

language

>Work lyric language as per ISO 639-3[67] if track relationships are enabled in Options and a related work exists. (*since Picard 0.10*)

lyrics

>Lyrics for the track.

media

>Release Format (e.g.: CD). See the Release Format[68] page on the MusicBrainz website for more information.

---

[66] https://en.wikipedia.org/wiki/International_Standard_Recording_Code
[67] https://en.wikipedia.org/wiki/ISO_639-3
[68] https://musicbrainz.org/doc/Release/Format

musicbrainz_albumartistid

Release Artist's MusicBrainz Identifier.

musicbrainz_albumid

Release MusicBrainz Identifier.

musicbrainz_artistid

Track Artist's MusicBrainz Identifier.

musicbrainz_discid

Disc ID if the album was added using *"Tools → Lookup CD"*. (*since Picard 0.12*)

musicbrainz_originalalbumid

Original Release's MusicBrainz Identifier.

musicbrainz_originalartistid

Original Track Artist's MusicBrainz Identifier.

musicbrainz_recordingid

Recording's MusicBrainz Identifier.

musicbrainz_releasegroupid

Release Group's MusicBrainz Identifier.

musicbrainz_releasetrackid

Release Track MusicBrainz Identifier. (*since Picard 1.3*)

musicbrainz_trackid

MusicBrainz Identifier for the track.

musicbrainz_workid

MusicBrainz Identifier for the work.

originalalbum

Release Title of the earliest release in the Release Group intended for the title of the original recording.

originalartist

Track Artist of the earliest release in the Release Group intended for the performer(s) of the original recording.

originaldate

Release Date (YYYY-MM-DD) of the earliest release in the Release Group intended to provide, for example, the release date of the vinyl version of what you have on CD. (*Included as standard from Picard 0.15, and using the Original Release Date plugin if you are still using a non-NGS version earlier than Picard 0.15*)

Note: If you are storing tags in MP3 files as ID3v2.3 (which is the Windows and iTunes compatible version) then the original date can only be stored as a year.

---

originalyear

Year of the original Release Date intended for release year of the original recording.

releasecountry

Country in which the release was issued.

releasestatus

Release Status indicating the "official" status of the release. Possible values include official, promotional, bootleg, and pseudo-release.

releasetype

Release Group Type (see also *_primaryreleasetype* and *_secondaryreleasetype*)

script

The script used to write the release's track list. The possible values are taken from the ISO 15924[69] standard. (*since Picard 0.10*)

subtitle

Used for information directly related to the contents title.

title

Track Title.

titlesort

Track Title's Sort Name.

totaldiscs

Total number of discs in this release

totaltracks

Total tracks on this disc.

tracknumber

Track number on the disc.

website

Used for official artist website.

---

[69] https://en.wikipedia.org/wiki/ISO_15924

5.2 Advanced Tags

You can make additional tags available by enabling the *Advanced Relationships* and the *Use genres from MusicBrainz* settings in Picard.

Note: Tags will not be created and will not be available as variables if there was no value retrieved for the tag from the MusicBrainz database.

Note: Some of these tags are only supported for certain file types or tag formats. Please see the Picard Tag Mapping[70] webpage for details.

5.2.1 Advanced Relationship Tags

If you enable tagging with Advanced Relationships, you get these extra tags:

arranger

Arranger Relationship Type (releases, recordings, works), Instrumentator Relationship Type, Orchestrator Relationship Type (*since Picard 0.10*)

composer

Composer Relationship Type

composersort

Composer Relationship Type's Sort Name

conductor

Conductor Relationship Type (releases, recordings), Chorus Master Relationship Type (releases, recordings)

djmixer

Mix-DJ Relationship Type (*since Picard 0.9*)

engineer

Engineer Relationship Type

license

License Relationship Type (releases, recordings) (*since Picard 1.0*)

lyricist

Lyricist Relationship Type

mixer

---

[70] https://picard.musicbrainz.org/docs/mappings/

Engineer Relationship Type ("Mixed By") (*since Picard 0.9*)

performer:<type>

Performer Relationship Type (releases - vocals/instruments, recordings - vocals/instruments), <type> can be "vocal", "guest guitar", "solo violin", . . .

Orchestra Relationship Type (releases, recordings), <type> is "orchestra"

Concertmaster Relationship Type (releases, recordings), <type> is "concertmaster"

producer

Producer Relationship Type

remixer

Remixer Relationship Type

work

Work Name (*since Picard 1.3*)

writer

Writer Relationship Type (*since Picard 1.0*). Not written to most file formats automatically. You can merge this with composers with a script like:

```
$copymerge(composer, writer)
```

5.2.2 Genre Tags

If you enable Use genres from MusicBrainz, you get:

genre

Genre information from MusicBrainz (*since Picard 2.1, earlier versions used folksonomy tags*)

5.3 Basic Variables

These variables are populated from MusicBrainz data for most releases, without any special Picard settings.

Note: Variables will not be created if there was no value retrieved for the variable from the MusicBrainz database.

_absolutetracknumber

The absolute number of this track disregarding the disc number (i.e.: `%_absolutetracknumber%` of `%_totalalbumtracks%`). For example, this value would be 11 for the second track on disc 2 where disc 1 has 9 tracks. (*since Picard 1.3*)

_albumartists

The Album's Artists' Name(s) (multi-value). (*since Picard 1.3*)

_albumartists_sort

The Album Artist's Sort Name(s) (multi-value). (*since Picard 1.3*)

_artists_sort

The Artist's Sort Name(s) (multi-value). (*since Picard 1.3*)

_bitrate

Approximate bitrate in kbps.

_bits_per_sample

Bits of data per sample.

_channels

Number of audio channels in the file.

_dirname

The name of the directory containing the file at the point of being loaded into Picard. (*since Picard 1.1*)

_extension

The file's extension. (*since Picard 0.9*)

_filename

The name of the file without extension (*since Picard 1.1*)

_format

Media format of the file (e.g.: MPEG-1 Audio).

_length

The length of the track in format mins:secs.

_multiartist

0 if tracks on the album all have the same primary artist, 1 otherwise. (*since Picard 1.3*)

_primaryreleasetype

Release Group Primary type (i.e.: album, single, ep, broadcast, or other).

_rating

Rating 0-5 by Musicbrainz users.

_releasecomment

Release disambiguation comment. (*since Picard 0.15*)

_releasegroup

Release Group Title which is typically the same as the Album Title, but can be different.

_releasegroupcomment

> Release Group disambiguation comment.

_releaselanguage

> Release Language as per ISO 639-3[71]. (*since Picard 0.10*)

_sample_rate

> Number of digitizing samples per second (Hz).

_secondaryreleasetype

> Zero or more Release Group Secondary types (i.e.: audiobook, compilation, dj-mix, interview, live, mixtape/street, remix, soundtrack, or spokenword).

_totalalbumtracks

> The total number of tracks across all discs of this release.

5.4 Advanced Variables

If you enable tagging with *Advanced Relationships*, you get these extra variables:

Note: Variables will not be created if there was no value retrieved for the variable from the MusicBrainz database.

_performance_attributes

> List of performance attributes for the work (e.g.: "live", "cover", "medley"). Use `$inmulti` to check for a specific type (i.e.: `$if($inmulti(%_performance_attributes%, medley), (Medley),))`. (*since Picard 1.3*)

_recordingcomment

> Recording disambiguation comment. (*since Picard 0.15*)

_recordingtitle

> Recording title - normally the same as the Track title, but can be different.

---

[71] https://en.wikipedia.org/wiki/ISO_639-3

5.5 Classical Music Tags

With the help of plugins like "Classical Extras" or "Work & Movement" you can make use of the following tags for tagging your classical music.

movement

>   Name of the movement (e.g.: "Andante con moto").

movementnumber

>   Movement number in Arabic numerals (e.g.: "2"). Players explicitly supporting this tag will often display it in Roman numerals (e.g.: "II").

movementtotal

>   Total number of movements in the work (e.g.: "4").

showmovement

>   Show Work & Movement: If this tag is set to "1" players supporting this tag, such as iTunes and MusicBee, will display the work, movement number and movement name instead of the track title. For example, the track will be displayed as "Symphony no. 5 in C minor, op. 67: II. Andante con moto" regardless of the value of the title tag.

work

>   Work Name of the overall work (e.g.: "Symphony no. 5 in C minor, op. 67").

Note: If you are using iTunes together with MP3 files you should activate the "Save iTunes compatible grouping and work" option in order for the work to be displayed correctly.

5.6 Tags from Plugins

Plugins from Picard *Plugins* can add more tags. Following are some examples.

5.6.1 Last.fm Plugin

genre

>   Pseudo-genre based on folksonomy tags.

5.6.2 Additional Artists Variables Plugin

Album Variables

_artists_album_primary_id

> The ID of the primary / first album artist listed

_artists_album_primary_std

> The primary / first album artist listed (standardized)

_artists_album_primary_cred

> The primary / first album artist listed (as credited)

_artists_album_primary_sort

> The primary / first album artist listed (sort name)

_artists_album_additional_id

> The IDs of all album artists listed except for the primary / first artist, as a multi-value

_artists_album_additional_std

> All album artists listed (standardized) except for the primary / first artist, separated by strings provided from the release entry

_artists_album_additional_cred

> All album artists listed (as credited) except for the primary / first artist, separated by strings provided from the release entry

_artists_album_additional_sort

> All album artists listed (sort names) except for the primary / first artist, separated by strings provided from the release entry

_artists_album_additional_std_multi

> All album artists listed (standardized) except for the primary / first artist, as a multi-value

_artists_album_additional_cred_multi

> All album artists listed (as credited) except for the primary / first artist, as a multi-value

_artists_album_all_std

> All album artists listed (standardized), separated by strings provided from the release entry

_artists_album_all_cred

> All album artists listed (as credited), separated by strings provided from the release entry

_artists_album_all_sort

All album artists listed (sort names), separated by strings provided from the release entry

_artists_album_all_std_multi

All album artists listed (standardized), as a multi-value

_artists_album_all_cred_multi

All album artists listed (as credited), as a multi-value

_artists_album_all_sort_primary

The primary / first album artist listed (sort name) followed by all additional album artists (standardized), separated by strings provided from the release entry

_artists_album_all_count

The number of artists listed as album artists

Track Variables

_artists_track_primary_id

The ID of the primary / first track artist listed

_artists_track_primary_std

The primary / first track artist listed (standardized)

_artists_track_primary_cred

The primary / first track artist listed (as credited)

_artists_track_primary_sort

The primary / first track artist listed (sort name)

_artists_track_additional_id

The IDs of all track artists listed except for the primary / first artist, as a multi-value

_artists_track_additional_std

All track artists listed (standardized) except for the primary / first artist, separated by strings provided from the track entry

_artists_track_additional_cred

All track artists listed (as credited) except for the primary / first artist, separated by strings provided from the track entry

_artists_track_additional_sort

All track artists listed (sort names) except for the primary / first artist, separated by strings provided from the track entry

_artists_track_additional_std_multi

All track artists listed (standardized) except for the primary / first artist, as a multi-value

_artists_track_additional_cred_multi

> All track artists listed (as credited) except for the primary / first artist, as a multi-value

_artists_track_all_std

> All track artists listed (standardized), separated by strings provided from the track entry

_artists_track_all_cred

> All track artists listed (as credited), separated by strings provided from the track entry

_artists_track_all_sort

> All track artists listed (sort names), separated by strings provided from the track entry

_artists_track_all_std_multi

> All track artists listed (standardized), as a multi-value

_artists_track_all_cred_multi

> All track artists listed (as credited), as a multi-value

_artists_track_all_sort_primary

> The primary / first track artist listed (sort name) followed by all additional track artists (standardized), separated by strings provided from the track entry

_artists_track_all_count

> The number of artists listed as track artists

Note: Some plugins make a large number of web service calls to get additional track-specific data such as performer and work relationships, so loading a large number of albums and tracks could take a significant amount of time. The time concern can be partially addressed by operating a local MusicBrainz server with the rate limiting disabled. Please see the MusicBrainz Server[72] project on GitHub for additional information.

5.7 Other Information

For technical details on how tags are written into files, see Picard Tag Mapping[73].

If you set new variables, these will be saved as new tags in ID3, APEv2 and VORBIS based files. For ID3 based files these will be saved to, and reloaded from, ID3 user defined text information (TXXX) frames. They will not be saved in ASF, MP4 or WAV based files.

---

[72] https://github.com/metabrainz/musicbrainz-server
[73] https://picard.musicbrainz.org/docs/mappings/

For ID3 based tags (i.e.: MP3 files), you can also set ID3 tags directly from your scripts by setting a special variable starting with "_id3:". Currently these tags are not loaded into variables when you reload the file into Picard. (*since Picard 0.9*)

SCRIPTING

Scripts are used to control some aspects of the operation of Picard.

There are two types of scripts used in Picard: the file naming script and tagging scripts. These are managed from the "File Naming" and "Scripting" sections of the *"Options → Options..."* menu.

Scripts are often discussed in the MetaBrainz Community Forum[74], and there is a thread specific to file naming and script snippets[75].

See also:

Please refer to the section on *Scripts* in *Extending Picard* for additional details about the two types of scripts, including how and when each of the scripts are executed.

6.1 Syntax

The syntax is derived from Foobar2000's titleformat[76]. There are three base elements: text, variable and function. Variables consist of alphanumeric characters enclosed in percent signs (e.g.: `%artist%`). Functions start with a dollar sign and end with an argument list enclosed in parentheses (e.g.: `$lower(...)`).

To use parentheses or commas as-is inside a function call, you must escape them with a backslash.

6.2 Metadata Variables

See *Tags & Variables* for the list of the variables provided by Picard.

Picard's variables can be either simple variables containing a single text string, or multi-value variables containing multiple text strings. In scripts, multi-value variables are automatically converted to a single text string by joining the values with a semicolon ";", except when used with special multi-value functions.

Note: The full list of available scripting functions is covered in the following chapter.

---

[74] https://community.metabrainz.org/
[75] https://community.metabrainz.org/t/repository-for-neat-file-name-string-patterns-and-tagger-script-snippets/2786/
[76] https://wiki.hydrogenaud.io/index.php?title=Foobar2000:Titleformat_Reference

SCRIPTING FUNCTIONS

The following is a list of the Picard scripting functions grouped by function type.

7.1 Assignment Functions

These functions are used to assign (or unassign) a value to a tag or variable. The assignment scripting functions are:

7.1.1 $copy

Usage: **$copy(target,source)**
Category: assignment
Implemented: Picard 0.9

Description:

Copies metadata from variable `source` to `target`. The difference from `$set(target,%source%)` is that `$copy(target,source)` copies multi-value variables without flattening them.

Note that if the variable `target` already exists, it will be overwritten by `source`.

Example:

The following statements will yield the values for `target` as indicated:

```
$set(source,)
$set(target,This will be overwritten)
$copy(target,source)                    ==>  ""

$set(source,one)
$copy(target,source)                    ==>  "one"

$setmulti(source,one)
$copy(target,source)                    ==>  "one"

$setmulti(source,one; two)
$copy(target,source)                    ==>  "one; two"
```

7.1.2 $copymerge

Usage: **$copymerge(target,source)**
Category: assignment
Implemented: Picard 1.0

Description:

Merges metadata from variable `source` into `target`, removing duplicates and appending to the end, so retaining the original ordering. Like `$copy`, this will also copy multi-valued variables without flattening them. Following the operation, `target` will be a multi-value variable.

Note that the variable names for `target` and `source` are passed directly without enclosing them in percent signs '%'.

Example:

The following statements will yield the values for `target` as indicated:

```
$set(target,)
$set(source,one)
$copymerge(target,source)      ==>   "one"

$set(target,zero)
$set(source,one)
$copymerge(target,source)      ==>   "zero; one"

$set(target,zero)
$setmulti(source,one; two)
$copymerge(target,source)      ==>   "zero; one; two"

$setmulti(target,zero; two)
$setmulti(source,one; two)
$copymerge(target,source)      ==>   "zero; two; one"

$set(target,zero; one; zero)
$set(source,one; two; three)
$copymerge(target,source)      ==>   "zero, one; two; three"
```

7.1.3 $delete

Usage: **$delete(name)**
Category: assignment
Implemented: Picard 2.1

Description:

Unsets the variable `name` and marks the tag for deletion.

This is similar to `$unset(name)` but also marks the tag for deletion. For example, running `$delete(genre)` will actually remove the "genre" tag from a file when saving.

Example:

The following statements will perform the actions indicated:

```
$delete(genre)  ==>  Remove the "genre" tag from a file when saving
```

7.1.4 $set

Usage: **$set(name,value)**
Category: assignment

Description:

Sets the variable `name` to `value`. The value of a variable is available to other script functions if it is enclosed between '%' characters (e.g.: `%name%`). If `name` is another variable (e.g.: `%indirect%`) the value of the variable will be used as `name`. This allows the creation of dynamically named variables.

Note: To create a variable which can be used for the file naming string, but which will not be written as a tag in the file, prefix the variable name with an underscore. `%something%` will create a "something" tag; `%_something%` will not.

Example:

The following statements will return the values indicated:

```
$set(comment,Testing)  ==>  "Testing" will be written to the "comment" tag
$set(_hidden,Testing)  ==>  "_hidden" variable will not be written

$set(_base,redirect)
$set(%_base%,Testing)  ==>  "Testing" will be written to the "redirect" tag
```

7.1.5 $setmulti

Usage: **$setmulti(name,value[,separator])**
Category: assignment
Implemented: Picard 1.0

Description:

Sets the variable `name` to `value`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the `value` back into a proper multi-valued variable. This can be used to operate on multi-valued variables as a string, and then set them back as proper multi-valued variable.

Example:

The following statements will return the values indicated:

```
$setmulti(genre,$lower(%genre%))   ==>  all "genre" elements in lower case
$setmulti(alpha,A; B; C)           ==>  3 elements ("A", "B" and "C")
$setmulti(mixed,A:A; B:B,:)         ==>  3 elements ("A", "A; B" and "B")
```

7.1.6 $unset

Usage: **$unset(name)**
Category: assignment

Description:

Unsets the variable `name`. The function allows for wildcards to unset certain tags (works with 'performer:*', 'comment:*', and 'lyrics:*').

Example:

The following would unset all performer tags:

```
$unset(performer:*)
```

7.2 Text Functions

These functions are used to manage text (e.g.: extract, replace or format) in tags or variables. The text scripting functions are:

7.2.1 $delprefix

Usage: **$delprefix(text[,prefixes])**
Category: text
Implemented: Picard 1.3

Description:

Deletes the specified `prefixes` from the beginning of `text`. Any number of `prefixes` can be specified, separated by commas. If no prefix is specified "A" and "The" are used by default. Note that the matching is case-sensitive.

Example:

The following statements will return the values indicated:

```
$delprefix(The Beatles)        ==>  "Beatles"
$delprefix(The Beatles,)       ==>  "The Beatles"
$delprefix(THE Beatles)        ==>  "THE Beatles"
$delprefix(THE Beatles,THE)    ==>  "Beatles"
$delprefix(The Beatles,A,An)   ==>  "The Beatles"
```

7.2.2 $find

Usage: **$find(haystack,needle)**
Category: text
Implemented: Picard 2.3

Description:

Returns the zero-based index of the first occurrance of `needle` in `haystack`, or an empty string if `needle` was not found. The comparisons are case-sensitive. If `needle` is blank, it will match the begging of `haystack` and return "0". The function does not support wildcards.

Note: Prior to Picard 2.3.2 `$find` returned "-1" if `needle` was not found.

Example:

The following statements will return the values indicated:

```
$find(abcdef,a)     ==>  "0"
$find(abcdef,c)     ==>  "2"
$find(abcdef,cd)    ==>  "2"
$find(abcdef,g)     ==>  ""
$find(abcdef,B)     ==>  ""
$find(,a)           ==>  ""
$find(abcdef,)      ==>  "1"
```

7.2.3 $firstalphachar

Usage: **$firstalphachar(text[,nonalpha])**
Category: text
Implemented: Picard 0.12

Description:

Returns the first character of `text` in upper case. If `text` does not begin with an alphabetic character, then `nonalpha` is returned instead. If `nonalpha` is not specified, the default value "#" will be used.

Example:

The following statements will return the values indicated:

```
$firstalphachar(abc)            ==>   "A"
$firstalphachar(123)            ==>   "#"
$firstalphachar(***)            ==>   "#"
$firstalphachar(***,)           ==>   ""
$firstalphachar(***,^)          ==>   "^"
$firstalphachar(***,non-alpha)  ==>   "non-alpha"
```

7.2.4 $firstwords

Usage: **$firstwords(text,length)**
Category: text
Implemented: Picard 0.12

Description:

Truncate `text` to `length`, only returning the complete words from `text` which fit within `length` characters. If `length` is less than 0, then the value used is the number of characters in `text` plus `length` (e.g.: `$firstwords(one two three,-3)` is the same as `$firstwords(one two three,10)`). If `length` is missing or a negative number greater than the number of characters in `text`, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$firstwords(Once upon a time,)      ==>   ""
$firstwords(Once upon a time,0)     ==>   ""
$firstwords(Once upon a time,3)     ==>   ""
$firstwords(Once upon a time,7)     ==>   "Once"
$firstwords(Once upon a time,-3)    ==>   "Once upon a"
$firstwords(Once upon a time,-30)   ==>   ""
```

7.2.5 $get

Usage: **$get(name)**
Category: text

Description:

Returns the variable `name` (equivalent to `%name%`) or an empty string if `name` has not been set. If `name` is another variable (e.g. `%indirect%`) the value of the variable will be used as `name`. This allows the retrieval of dynamically named variables.

Example:

The following statements will return the values indicated:

```
$set(foo,This is foo)
$set(bar,foo)
$get(foo)              ==>  "This is foo"
$get(bar)              ==>  "foo"
$get(%bar%)            ==>  "This is foo"
$get(baz)              ==>  "" ('baz' has not been set to a value)
```

7.2.6 $initials

Usage: **$initials(text)**

Category: text

Implemented: Picard 0.12

Description:

Returns the first character of each word in `text`, if it is an alphabetic character.

Example:

The following statements will return the values indicated:

```
$set(foo,This is a test)
$initials(%foo%)             ==>  "Tiat"
$initials(This is a test)    ==>  "Tiat"
$initials(This is a 123 test)  ==>  "Tiat"
```

7.2.7 $left

Usage: **$left(text,num)**

Category: text

Description:

Returns the first `num` characters from `text`. If `num` is less than 0, then the value used is the number of characters in `text` plus `num` (e.g.: `$right(abcd,-1)` is the same as `$right(abcd,3)`). If `num` is missing or a negative number greater than the number of characters in `text`, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$left(,)       ==>  ""
$left(ABC,)    ==>  ""
$left(ABC,0)   ==>  ""
$left(ABC,2)   ==>  "AB"
$left(ABC,4)   ==>  "ABC"
```

(continues on next page)

a

7.2.10 $num

Usage: **$num(num,len)**
Category: text

Description:

Returns `num` formatted to `len` digits, where `num` and `len` are integers and `len` cannot be greater than 20.

Example:

The following statements will return the values indicated:

```
$num(,)          ==>   ""
$num(,1)         ==>   "0"
$num(a,)         ==>   ""
$num(a,5)        ==>   "00000"
$num(123,5)      ==>   "00123"
$num(1.23,5)     ==>   "00000"
$num(123,)       ==>   ""
$num(123,0)      ==>   "123"
$num(123,1)      ==>   "123"
$num(123,20)     ==>   "00000000000000000123"
$num(123,50)     ==>   "00000000000000000123"
$num(123,5.5)    ==>   ""
$num(1.23,10)    ==>   "0000000000"
```

7.2.11 $pad

Usage: **$pad(text,length,char)**
Category: text

Description:

Pads the `text` to the `length` provided by adding as many copies of `char` as needed to the beginning of the string. For the padded length to be correct, `char` must be exactly one character in length. If `length` is less than the number of characters in `text`, the function will return `text`. If `length` is missing or is not a number, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$pad(abc,5,+)    ==>   "++abc"
$pad(abc,0,+)    ==>   "abc"
$pad(abc,5,)     ==>   "abc"
$pad(abc,5,XY)   ==>   "XYXYabc"   (note final length is incorrect)
$pad(abc,,+)     ==>   ""
$pad(abc,x,+)    ==>   ""
```

7.2.12 $replace

Usage: **$replace(text,search,replace)**
Category: text

Description:

Replaces occurrences of `search` in `text` with `replace` and returns the resulting string.

Example:

The following statements will return the values indicated:

```
$set(foo,I like cats the best)
$replace(%foo%,cat,dog)                ==>  "I like dogs the best"

$set(foo,I like cats the best)
$set(bar,cat)
$replace(%foo%,%bar%,dog)              ==>  "I like dogs the best"

$set(foo,I like cats the best)
$set(bar,cat)
$set(baz,dog)
$replace(%foo%,%bar%,%baz%)            ==>  "I like dogs the best"

$replace(I like cats the best,cat,dog)  ==>  "I like dogs the best"
$replace(I like cats the best,pig,dog)  ==>  "I like cats the best"
$replace(I like cats the best,cat,)     ==>  "I like s the best"
$replace(Bad replace,,_)                ==>  "_B_a_d_ _r_e_p_l_a_c_e_"
```

7.2.13 $reverse

Usage: **$reverse(text)**
Category: text

Description:

Returns `text` in reverse order.

Example:

The following statements will return the values indicated:

```
$set(foo,abcde)
$reverse(%foo%)  ==>  "edcba"

$reverse(abcde)  ==>  "edcba"
```

7.2.14 $right

Usage: **$right(text,num)**
Category: text

Description:

Returns the last `num` characters from `text`. If `num` is less than 1, then the value used is the number of characters in `text` plus num (e.g.: `$right(abcd,0)` is the same as `$right(abcd,4)`). If `num` is missing or a negative number greater than the number of characters in `text`, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$right(abcd,2)   ==>  "cd"
$right(abcd,0)   ==>  "cd"
$right(abcd,-1)  ==>  "bcd"
$right(abcd,)    ==>  ""
$right(abcd,-5)  ==>  ""
```

7.2.15 $rreplace

Usage: **$rreplace(text,pattern,replace)**
Category: text

Description:

Regular expression replace. This function will replace the matching group specified by `pattern` with `replace` in `text`. For more information about regular expressions, please see the article on Wikipedia[77].

Example:

The following statements will return the values indicated:

```
$rreplace(test \(disc 1\),\\s\\\(disc \\d+\\\),)  ==>  "test"
$rreplace(test,[t,)                               ==>  "test"
```

---

[77] https://wikipedia.org/wiki/Regular_expression

7.2.16 **$rsearch**

Usage: **$rsearch(text,pattern)**

Category: text

Description:

Regular expression search. This function will return the first matching group specified by `pattern` from `text`. For more information about regular expressions, please see the [article on Wikipedia](https://wikipedia.org/wiki/Regular_expression)[78].

Example:

The following statements will return the values indicated:

```
$rsearch(test \(disc 1\),\\\(disc \(\\d+\)\\\))   ==>   "1"
$rsearch(test \(disc 1\),\\\(disc \\d+\\\))       ==>   "(disc 1)"
$rsearch(test,x)                                  ==>   ""
```

7.2.17 **$strip**

Usage: **$strip(text)**

Category: text

Description:

Replaces all whitespace in `text` with a single space, and removes leading and trailing spaces. Whitespace characters include multiple consecutive spaces, and various other unicode characters. Characters such as newlines '\n', tabs '\t' and returns '\r' are treated as spaces.

Example:

The following statements will each return "This text has been stripped.":

```
$strip(This text has been stripped.)
$strip(This text has been stripped.  )
$strip(  This text has been stripped.  )
$strip(  This  text has been stripped.)
$strip(  This  text has been stripped.  )
$strip(This  text  has  been  stripped.)
$strip(This  text\rhas\nbeen\tstripped.)
```

---

[78] https://wikipedia.org/wiki/Regular_expression

7.2.18 $substr

Usage: **$substr(text,start,end)**
Category: text
Implemented: Picard 2.3

Description:

Returns the substring of `text` beginning with the character at the `start` index, up to (but not including) the character at the `end` index. Indexes are zero-based. Negative numbers will be counted back from the end of the string. If the start or end indexes are left blank, they will default to the start and end of the string respectively. If the `start` index evaluates to a negative number (e.g.: `text` is "abc" and `start` is -10), it will default to the start of the string. Similarly, if `end` index is a number greater than the number of characters in the string, it will default to the end of the string. Invalid index values (e.g.: `start` greater than `end`) will return an empty string.

Example:

The following statements will return the values indicated:

```
$substr(abcdefg)        ==>  "abcdefg"
$substr(abcdefg,3)      ==>  "defg"
$substr(abcdefg,,3)     ==>  "abc"
$substr(abcdefg,0,3)    ==>  "abc"
$substr(abcdefg,-3)     ==>  "efg"
$substr(abcdefg,-6,3)   ==>  "bc"
$substr(abcdefg,-10,3)  ==>  "abc"
$substr(abcdefg,3,1)    ==>  ""
```

7.2.19 $swapprefix

Usage: **$swapprefix(text[,prefixes])**
Category: text
Implemented: Picard 1.3 (*previously as a plugin since Picard 0.13*)

Description:

Moves the specified `prefixes` from the beginning to the end of `text`. Any number of `prefixes` can be specified, separated by commas. If no prefix is specified "A" and "The" are used by default. Note that the matching is case-sensitive.

Example:

If the `albumartist` is "Le Butcherettes", the following statements will return the values indicated:

```
$swapprefix(%albumartist%)            ==>  "Le Butcherettes"
$swapprefix(%albumartist%,le)         ==>  "Le Butcherettes"
```

```
$swapprefix(%albumartist%,L)          ==>   "Le Butcherettes"
$swapprefix(%albumartist%,A,An,The,Le)  ==>  "Butcherettes, Le"
```

7.2.20 $title

Usage: **$title(text)**

Category: text

Implemented: Picard 2.1

Description:

Returns `text` in title case (first character in every word capitalized).

Example:

The following statement will return the value indicated:

```
$title(tHe houR is upOn uS)  ==>  "The Hour Is Upon Us"
```

7.2.21 $trim

Usage: **$trim(text[,char])**

Category: text

Description:

Trims all leading and trailing whitespaces from `text`. The optional second parameter `char` specifies the character to trim. If multiple characters are provided in `char`, each character will be applied separately to the function.

Examples:

The following statements will return the values indicated:

```
$trim(  Trimmed  )       ==>  "Trimmed"
$trim(__Trimmed__,_)     ==>  "Trimmed"
$trim(x__Trimmed__y,_x)  ==>  "Trimmed__y"
```

7.2.22 $truncate

Usage: **$truncate(text,length)**
Category: text
Implemented: Picard 0.12

Description:

Truncate `text` to `length`. If `length` is less than 0, then the value used is the number of characters in `text` plus `length` (e.g.: `$truncate(abcd,-1)` is the same as `$truncate(abcd,3)`). If `length` is missing or a negative number greater than the number of characters in `text`, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$truncate(Once upon a time,)     ==>   ""
$truncate(Once upon a time,0)    ==>   ""
$truncate(Once upon a time,3)    ==>   "Onc"
$truncate(Once upon a time,-3)   ==>   "Once upon a t"
$truncate(Once upon a time,-30)  ==>   ""
```

7.2.23 $upper

Usage: **$upper(text)**
Category: text

Description:

Returns `text` in upper case.

Example:

The following statement will return the value indicated:

```
$upper(This text is UPPER case)  ==>   "THIS TEXT IS UPPER CASE"
```

7.3 Multi-Value Functions

These functions are used to manage multi-value tags or variables. The multi-value scripting functions are:

7.3.1 $getmulti

Usage: **$getmulti(name,index[,separator])**
Category: multi-value
Implemented: Picard 2.3

Description:

Gets the element at `index` from the multi-value variable `name`. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-value variable.

The `index` is zero based. If `index` is less than 0, then the value used is the number of elements in `name` plus `index` (e.g.: `$getmulti(%abcd%,-1)` is the same as `$getmulti(%abcd%,3)` if `%abcd%` is a multi-value variable with four elements). If `index` is missing, not an integer, a number greater than or equal to the number of elements in `name`, or a negative number greater than the number of elements in `name`, then the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$set(foo,A; B; C)
$setmulti(bar,A; B; C)
$set(baz,1)
$getmulti(%foo%,%baz%)        ==>  ""
$getmulti(%foo%,0)            ==>  "A; B; C"
$getmulti(%foo%,-1)           ==>  "A; B; C"
$getmulti(%foo%,-%baz%)       ==>  "A; B; C"
$getmulti(%bar%,%baz%)        ==>  "B"
$getmulti(%bar%,0)            ==>  "A"
$getmulti(%bar%,-1)           ==>  "C"
$getmulti(%bar%,-%baz%)       ==>  "C"

$getmulti(A:1; B:2; C:3,1)    ==>  "B:2"
$getmulti(A:1; B:2; C:3,1,:)  ==>  "1; B"
$getmulti(A:1; B:2; C:3,10)   ==>  ""
$getmulti(A:1; B:2; C:3,-10)  ==>  ""
$getmulti(A:1; B:2; C:3,1.5)  ==>  ""
$getmulti(A:1; B:2; C:3,a)    ==>  ""
```

7.3.2 $join

Usage: **$join(name,text[,separator])**
Category: multi-value
Implemented: Picard 2.3

Description:

Joins all elements in the multi-value variable `name`, placing `text` between each element, and returns the result as a string. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable.

Example:

The following statements will return the values indicated:

```
$set(foo,First:A; Second:B)
$join(%foo%, >> )                  ==>  "First:A; Second:B"
$join(%foo%, >> ,:)                ==>  "First >> A; Second >> B"

$setmulti(bar,First:A; Second:B)
$join(%bar%, >> )                  ==>  "First:A >> Second:B"
$join(%bar%, >> ,:)                ==>  "First >> A; Second >> B"

$join(First:A; Second:B,)          ==>  "First:ASecond:B"
$join(First:A; Second:B, >> )      ==>  "First:A >> Second:B"
$join(First:A; Second:B, >> ,:)    ==>  "First >> A; Second >> B"
```

7.3.3 $lenmulti

Usage: **$lenmulti(name[,separator])**
Category: multi-value

Description:

Returns the number of elements in the multi-value variable `name`. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable. If `name` is missing $lenmulti will return "0". If `separator` is specified but left blank (e.g. $setmulti(A; B; C,)) the function will return "1".

Example:

The following statements will return the values indicated:

```
$set(foo,)
$lenmulti(%foo%)            ==>   "0"
```

```
$set(foo,A; B; C)
$lenmulti(%foo%)               ==>   "1"

$setmulti(foo,A; B; C)
$lenmulti(%foo%)               ==>   "3"

$lenmulti(A; B; C)             ==>   "3"
$lenmulti(A:A; B:B; C:C,:)     ==>   "4"
$lenmulti(,)                   ==>   "0"
$lenmulti(,:)                  ==>   "0"
$lenmulti(A; B; C,)            ==>   "1"
```

7.3.4 $map

Usage: **$map(name,code[,separator])**
Category: multi-value
Implemented: Picard 2.3

Description:

Iterates over each element found in the multi-value variable `name` and updates the value of the element to the value returned by `code`, returning the updated multi-value variable. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable.

For each loop, the element value is first stored in the variable `_loop_value` and the count is stored in the variable `_loop_count`. This allows the element or count value to be accessed within the code script.

Note: You cannot save the `code` to a variable and then pass the variable to the function as `%code%` because it will be evaluated when it is assigned to the variable rather than during the loop.

Example:

The following statements will return the values indicated:

```
$set(foo,First:A; Second:B)
$map(%foo%,
    $upper(%_loop_count%=%_loop_value%))    ==>   "1=FIRST:A; SECOND:B"
$map(%foo%,
    $upper(%_loop_count%=%_loop_value%),:)  ==>   "1=FIRST:2=A; SECOND:3=B"

$setmulti(bar,First:A; Second:B)
$map(%bar%,
    $upper(%_loop_count%=%_loop_value%))    ==>   "1=FIRST:A; 2=SECOND:B"
$map(%bar%,
    $upper(%_loop_count%=%_loop_value%),:)  ==>   "1=FIRST:2=A; SECOND:3=B"
```

```
$map(First:A; Second:B,
    $upper(%_loop_count%=%_loop_value%))    ==>   "1=FIRST:A; 2=SECOND:B"
```

7.3.5 $performer

Usage: **$performer(pattern[,separator])**
Category: multi-value
Implemented: Picard 0.10

Description:

Returns the performers where the performance type matches `pattern` separated by `separator` (or a comma followed by a space ", " if not passed). If `pattern` is blank, then all performers will be returned. Note that the `pattern` to be matched is case-sensitive.

Example:

With the performer tags as `performer:guitar` = "Ann", `performer:rhythm-guitar` = "Bob" and `performer:drums` = "Cindy", the following statements will return the values indicated:

```
$set(foo,guitar)
$performer(%foo%)          ==>   "Ann, Bob"

$performer(guitar)         ==>   "Ann, Bob"
$performer(Guitar)         ==>   ""
$performer(rhythm-guitar)  ==>   "Bob"
$performer()               ==>   "Ann, Bob, Cindy"
$performer(, / )           ==>   "Ann / Bob / Cindy"
```

7.3.6 $reversemulti

Usage: **$reversemulti(name[,separator])**
Category: multi-value
Implemented: Picard 2.3.1

Description:

Returns a copy of the multi-value variable `name` with the elements in reverse order. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable.

This function can be used in conjunction with the `$sortmulti` function to sort in descending order.

Example:

The following statements will return the values indicated:

```
$set(foo,A; B; C; D; E)
$reversemulti(%foo%)            ==>  "A; B; C; D; E"

$setmulti(bar,A; B; C; D; E)
$reversemulti(%bar%)            ==>  "E; D; C; B; A"

$setmulti(baz,A:A; B:B; C:C,:)
$reversemulti(%baz%)            ==>  "C; B; C; A; B; A"

$reversemulti(A; B; C; D; E)    ==>  "E; D; C; B; A"
$reversemulti(A:A; B:B; C:C,:)  ==>  "C:B; C:A; B:A"
```

7.3.7 $slice

Usage: **$slice(name,start,end[,separator])**
Category: multi-value
Implemented: Picard 2.3

Description:

Returns a multi-value variable containing the elements from the `start` index up to but not including the `end` index from the multi-value variable `name`. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable.

Indexes are zero based. Negative numbers will be counted back from the number of elements in the list. If the `start` or `end` indexes are left blank, they will default to 0 and the number of elements in the list respectively.

A typical use might be to create a multi-value variable with all artists in `%artists%` except the first, which can be used to create a "feat." list. This would look something like `$setmulti(supporting_artists,$slice(%artists%,1,))`.

Example:

The following statements will return the values indicated:

```
$set(foo,A; B; C; D; E)
$slice(%foo%,1)                      ==>  ""

$setmulti(foo,A; B; C; D; E)
$slice(%foo%,1)                      ==>  "B; C; D; E"

$slice(A; B; C; D; E,1)              ==>  "B; C; D; E"
$slice(A; B; C; D; E,1,3)            ==>  "B; C"
$slice(A; B; C; D; E,,3)             ==>  "A; B; C"
$slice(A; B; C; D; E,1,3)            ==>  "B; C"
$slice(A; B; C; D; E,1,-1)           ==>  "B; C; D"
$slice(A; B; C; D; E,-4,4)           ==>  "B; C; D"
```

```
$slice(A:A; B:B; C:C; D:D; E:E,,1,:)   ==>   "A"
$slice(A:A; B:B; C:C; D:D; E:E,-2,,:)  ==>   "D; E:E"
$slice(A:A; B:B; C:C; D:D; E:E,2,4,:)  ==>   "B; C:C; D"
```

7.3.8 $sortmulti

Usage: **$sortmulti(name[,separator])**
Category: multi-value
Implemented: Picard 2.3.1

Description:

Returns a copy of the multi-value variable `name` with the elements sorted in ascending order. A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable. If `name` is missing `$sortmulti` will return an empty string.

Example:

The following statements will return the values indicated:

```
$set(foo,B; C; E; D; A)
$sortmulti(%foo%)                      ==>   "B; C; E; D; A"

$setmulti(foo,B; C; E; D; A)
$sortmulti(%foo%)                      ==>   "A; B; C; D; E"

$sortmulti(B; D; E; A; C)              ==>   "A; B; C; D; E"
$sortmulti(B:AB; D:C; E:D; A:A; C:X,:) ==>   "A; C:AB; D:B:C; E:D; A:X"
$sortmulti(,)                          ==>   ""
$sortmulti(,:)                         ==>   ""
```

7.4 Mathematical Functions

These functions are used to perform arithmetic operations on tags or variables. The mathematical scripting functions are:

7.4.1 $add

Usage: **$add(x,y,*args)**

Category: mathematical

Description:

Adds `y` to `x`. Can be used with an arbitrary number of arguments (i.e.: `$add(x,y,z)` = (x + y) + z). Returns an empty string if an argument is missing or not an integer.

Example:

The following statements will return the values indicated:

```
$add(20,15)      ==>   "35"
$add(20,-15)     ==>   "5"
$add(20,14,1)    ==>   "35"
$add(20,10,3,2)  ==>   "35"
$add(20,10,3,)   ==>   ""
$add(20,10,3,a)  ==>   ""
$add(20,10,3.5)  ==>   ""
```

7.4.2 $div

Usage: **$div(x,y,*args)**

Category: mathematical

Description:

Divides x by y and returns the integer value (rounded down). Can be used with an arbitrary number of arguments (i.e.: `$div(x,y,z)` = (x / y) / z). If an argument is empty or not an integer, the function will return an empty string. If the second or any subsequent argument is zero, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$div(10,3)      ==> "3"
$div(10,-3)     ==> "-4"
$div(-10,3)     ==> "-4"
$div(10,3,2)    ==> "1"
```

```
$div(10,-3,-2)  ==> "2"
$div(10,2,1.5)  ==> ""
$div(10,2,0)    ==> ""
$div(10,2,x)    ==> ""
$div(10,2,)     ==> ""
```

7.4.3 $mod

Usage: **$mod(x,y,*args)**

Category: mathematical

Description:

Returns the remainder of x divided by y. Can be used with an arbitrary number of arguments (i.e.: `$mod(x, y,z)` = (x % y) % z). If an argument is empty or not an integer, the function will return an empty string. If the second or any subsequent argument is zero, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$mod(0,3)      ==>  "0"
$mod(10,3)     ==>  "1"
$mod(10,-3)    ==>  "-2"
$mod(-13,10)   ==>  "7"
$mod(13,-10)   ==>  "-7"
$mod(10,3,1)   ==>  "0"
$mod(50,17,9)  ==>  "7"
$mod(51,3,0)   ==>  ""
$mod(51,a)     ==>  ""
$mod(a,10)     ==>  ""
$mod(,10)      ==>  ""
$mod(10,)      ==>  ""
$mod(10,3.5)   ==>  ""
```

7.4.4 $mul

Usage: **$mul(x,y,*args)**

Category: mathematical

Description:

Multiplies x by y. Can be used with an arbitrary number of arguments (i.e.: `$mul(x,y,z)` = (x * y) * z). If an argument is empty or not an integer, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$mul(1,2)        ==>  "2"
$mul(1,2,3)      ==>  "6"
$mul(1,2,0)      ==>  "0"
$mul(1,-2,3)     ==>  "-6"
$mul(-1,2,-3)    ==>  "6"
$mul(1,2,)       ==>  ""
$mul(1,2,x)      ==>  ""
$mul(1,2.5)      ==>  ""
```

7.4.5 $sub

Usage: **$sub(x,y,*args)**
Category: mathematical

Description:

Subtracts y from x. Can be used with an arbitrary number of arguments (i.e.: $\mathtt{\$sub(x,y,z)} = (x - y) - z$).
Returns an empty string if an argument is missing or not an integer.

Example:

The following statements will return the values indicated:

```
$sub(20,15)      ==>   "5"
$sub(20,-15)     ==>   "35"
$sub(20,14,1)    ==>   "5"
$sub(20,10,3,2)  ==>   "5"
$sub(20,10,3,)   ==>   ""
$sub(20,10,3,a)  ==>   ""
$sub(20,10,3.5)  ==>   ""
```

7.5 Conditional Functions

These functions are used to test for various conditions and take appropriate actions depending on the results of the test. The conditional scripting functions are:

7.5.1 $and

Usage: **$and(x,y,*args)**
Category: conditional

Description:

Returns true if both x and y are not empty. Can be used with an arbitrary number of arguments. The result is true if **ALL** of the arguments are not empty.

Example:

The following statements will return the values indicated:

```
$set(test,x)
$and(%test%,)          ==>  ""   (False)
$and(%test%,1)         ==>  "1"  (True)
$and(%test%,A)         ==>  "1"  (True)
$and(%test%,$gt(4,5))  ==>  ""   (False)
$and(%test%,$lt(4,5))  ==>  "1"  (True)
$and(%test%,,)         ==>  ""   (False)
$and(%test%,,0)        ==>  ""   (False)
$and(%test%,, )        ==>  ""   (False)
$and(%test%, , )       ==>  "1"  (True)
```

7.5.2 $endswith

Usage: **$endswith(text,suffix)**
Category: conditional
Implemented: Picard 1.4

Description:

Returns true if text ends with suffix. Note that the comparison is case-sensitive.

Example:

The statements below return the values indicated:

```
$endswith(The time is now,is now)  ==>  "1"  (True)
$endswith(The time is now,is NOW)  ==>  ""   (False)
$endswith(The time is now,)        ==>  "1"  (True)
$endswith(,)                       ==>  "1"  (True)
$endswith(,now)                    ==>  ""   (False)
```

7.5.3 $eq

Usage: **$eq(x,y)**
Category: conditional
Implemented: Picard

Description:

Returns true if x equals y. Note that comparisons are case-sensitive.

Example:

The following statements will return the values indicated:

```
$eq(,a)    ==>    ""    (False)
$eq(a,)    ==>    ""    (False)
$eq(a,A)   ==>    ""    (False)
$eq(a,a)   ==>    "1"   (True)
```

7.5.4 $eq_all

Usage: **$eq_all(x,a1,a2,*args)**

Category: conditional

Description:

Returns true if x equals a1 and a2, etc. Can be used with an arbitrary number of arguments. Note that comparisons are case-sensitive.

Functionally equivalent to $and($eq(x,a1),$eq(x,a2) ...).

Example:

The following statements will return the values indicated:

```
$eq_all(A,A,B,C)    ==>    ""    (False)
$eq_all(A,a,A,A)    ==>    ""    (False)
$eq_all(A,A,A,A)    ==>    "1"   (True)
$eq_all(,,,)        ==>    "1"   (True)
$eq_all(,a,)        ==>    ""    (False)
```

7.5.5 $eq_any

Usage: **$eq_any(x,a1,a2,*args)**

Category: conditional

Description:

Returns true if x equals a1 or a2, etc. Can be used with an arbitrary number of arguments. Note that comparisons are case-sensitive.

Functionally equivalent to $or($eq(x,a1),$eq(x,a2) ...).

Example:

The following statements will return the values indicated:

```
$eq_any(A,A,B,C)  ==>  "1"  (True)
$eq_any(A,a,A,A)  ==>  "1"  (True)
$eq_any(A,a,b,c)  ==>  ""   (False)
$eq_any(,,,)      ==>  "1"  (True)
$eq_any(,a,b,c)   ==>  ""   (False)
```

7.5.6 $gt

Usage: **$gt(x,y)**
Category: conditional
Implemented: Picard

Description:

Returns true if x is greater than y. If an argument is missing or is not an integer, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$gt(-1,0)   ==>   ""   (False)
$gt(0,0)    ==>   ""   (False)
$gt(1,0)    ==>   "1"  (True)
$gt(,)      ==>   ""   (False)
$gt(,0)     ==>   ""   (False)
$gt(0,)     ==>   ""   (False)
$gt(a,1)    ==>   ""   (False)
$gt(1,a)    ==>   ""   (False)
$gt(1,1.5)  ==>   ""   (False)
```

7.5.7 $gte

Usage: **$gte(x,y)**
Category: conditional
Implemented: Picard

Description:

Returns true if x is greater than or equal to y. If an argument is missing or is not an integer, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$gte(-1,0)   ==>  ""   (False)
$gte(0,0)    ==>  "1"  (True)
$gte(1,0)    ==>  "1"  (True)
$gte(,)      ==>  ""   (False)
$gte(,0)     ==>  ""   (False)
$gte(0,)     ==>  ""   (False)
$gte(a,1)    ==>  ""   (False)
$gte(1,a)    ==>  ""   (False)
$gte(1,1.5)  ==>  ""   (False)
```

7.5.8 $if

Usage: **$if(condition,then[,else])**
Category: conditional

Description:

If `condition` is not empty it returns `then`, otherwise it returns `else`. If `else` is not provided, it will be assumed to be an empty string. In addition to (or instead of) returning values, `then` and `else` can be used to conditionally execute other functions.

Example:

The following statements will return the values indicated:

```
$set(foo,This is foo)
$set(bar,)
$if(%foo%,%foo%,No foo)                   ==>  "This is foo"
$if(%bar%,%bar%,No bar)                   ==>  "No bar"
$if(%bar%,This is bar,No bar)             ==>  "No bar"
$if(%bar%,This is bar,)                    ==>  ""
$if(%bar%,This is bar)                     ==>  ""
$if(,True,False)                           ==>  "False"
$if( ,True,False)                          ==>  "True"
$if(,$set(value,True),$set(value,False))  ==>  Sets "value" to "False"
$set(value,$if(%bar%,True,False))          ==>  Sets "value" to "False"
```

7.5.9 $if2

Usage: **$if2(a1,a2,a3,…)**
Category: conditional

Description:

Returns the first non empty argument. Can be used with an arbitrary number of arguments.

Example:

The following statements will return the values indicated:

```
$set(foo,)
$set(bar,Something)
$if2(%foo%,%bar%,Three)     ==>  "Something"
$if2(,%bar%,Three)          ==>  "Something"
$if2(,%foo%,%bar%,Three)    ==>  "Something"
$if2(%foo%, ,%bar%,Three)   ==>  " "
$if2(%foo%.,%bar%,Three)    ==>  "."
$if2(%foo%,,Three)          ==>  "Three"
$if2(%foo%,,,)              ==>  ""
```

7.5.10 $in

Usage: **$in(x,y)**
Category: conditional
Implemented: Picard

Description:

Returns true, if x contains y. Note that comparisons are case-sensitive.

Example:

The following statements will return the values indicated:

```
$set(foo,ABCDEFG)
$set(bar,CDE)
$in(%foo%,%bar%)  ==>  "1"  (True)
$in(ABCDE,CDE)    ==>  "1"  (True)
$in(ABCDE,CE)     ==>  ""   (False)
$in(ABCDE,cde)    ==>  ""   (False)
$in(ABCDE,)       ==>  "1"  (True)
$in(,)            ==>  "1"  (True)
```

7.5.11 $inmulti

Usage: **$inmulti(%x%,y)**
Category: conditional
Implemented: Picard 1.0

Description:

Returns true if multi-value variable x contains exactly y as one of its values. Note that comparisons are case-sensitive.

Example:

The following statements will return the values indicated:

```
$setmulti(foo,One; Two; Three)
$set(bar,Two)
$inmulti(%foo%,%bar%)  ==>  "1"  (True)
$inmulti(%foo%,Two)    ==>  "1"  (True)
$inmulti(%foo%,two)    ==>  ""   (False)
$inmulti(%foo%,Once)   ==>  ""   (False)
$inmulti(%foo%,w)      ==>  ""   (False)
$inmulti(%foo%,)       ==>  ""   (False)
```

7.5.12 $is_audio

Usage: **$is_audio()**
Category: conditional
Implemented: Picard 2.2

Description:

Returns true, if the track being processed is not shown as being a video.

Example:

The following statements will return the values indicated:

```
$is_audio()  ==>  "1"  (True, if the track is not a video)
$is_audio()  ==>  ""   (False, if the track is a video)
```

7.5.13 $is_complete

Usage: **$is_complete()**
Category: conditional

Description:

Returns true if every track in the album is matched to a single file.

Note: This function only works in File Naming scripts.

Example:

The following statements will return the values indicated:

```
$is_complete()  ==>  "1"  (True, if all tracks have been matched)
$is_complete()  ==>  ""   (False, if not all tracks have been matched)
```

7.5.14 $is_video

Usage: **$is_video()**
Category: conditional
Implemented: Picard 2.2

Description:

Returns true, if the track being processed is shown as being a video.

Example:

The following statements will return the values indicated:

```
$is_video()  ==>  "1"  (True, if the track is a video)
$is_video()  ==>  ""   (False, if the track is not a video)
```

7.5.15 $lt

Usage: **$lt(x,y)**
Category: conditional

Description:

Returns true if x is less than y. If an argument is missing or is not an integer, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$lte(-1,0)   ==>  "1"  (True)
$lte(0,0)    ==>  ""   (False)
$lte(1,0)    ==>  ""   (False)
$lte(,)      ==>  ""   (False)
$lte(,0)     ==>  ""   (False)
$lte(0,)     ==>  ""   (False)
$lte(a,1)    ==>  ""   (False)
$lte(1,a)    ==>  ""   (False)
$lte(1,1.5)  ==>  ""   (False)
```

7.5.16 $lte

Usage: **$lte(x,y)**

Category: conditional

Description:

Returns true if x is less than or equal to y. If an argument is missing or is not an integer, the function will return an empty string.

Example:

The following statements will return the values indicated:

```
$lte(-1,0)    ==>   "1"   (True)
$lte(0,0)     ==>   "1"   (True)
$lte(1,0)     ==>   ""    (False)
$lte(,)       ==>   ""    (False)
$lte(,0)      ==>   ""    (False)
$lte(0,)      ==>   ""    (False)
$lte(a,1)     ==>   ""    (False)
$lte(1,a)     ==>   ""    (False)
$lte(1,1.5)   ==>   ""    (False)
```

7.5.17 $ne

Usage: **$ne(x,y)**

Category: conditional

Description:

Returns true if x does not equal y. Note that comparisons are case-sensitive.

Example:

The following statements will return the values indicated:

```
$ne(,a)    ==>   "1"   (True)
$ne(a,)    ==>   "1"   (True)
$ne(a,A)   ==>   "1"   (True)
$ne(a,a)   ==>   ""    (False)
```

7.5.18 $ne_all

Usage: **$ne_all(x,a1,a2,*args)**

Category: conditional

Description:

Returns true if x does not equal a1 and a2, etc. Can be used with an arbitrary number of arguments. Note that comparisons are case-sensitive.

Functionally equivalent to $and($ne(x,a1),$ne(x,a2) ...).

Example:

The following statements will return the values indicated:

```
$ne_all(A,A,B,C)    ==>  ""   (False)
$ne_all(A,a,A,A)    ==>  ""   (False)
$ne_all(A,a,a,a)    ==>  "1"  (True)
$ne_all(,,,)        ==>  ""   (False)
$ne_all(,a,a)       ==>  "1"  (True)
```

7.5.19 $ne_any

Usage: **$ne_any(x,a1,a2,*args)**

Category: conditional

Description:

Returns true if x does not equal a1 or a2, etc. Can be used with an arbitrary number of arguments. Note that comparisons are case-sensitive.

Functionally equivalent to $or($ne(x,a1),$ne(x,a2) ...).

Example:

The following statements will return the values indicated:

```
$ne_any(A,A,B,C)    ==>  "1"  (True)
$ne_any(A,a,A,A)    ==>  "1"  (True)
$ne_any(A,A,A,A)    ==>  ""   (False)
$ne_any(,,,)        ==>  ""   (False)
$ne_any(,a,,)       ==>  "1"  (True)
```

7.5.20 $not

Usage: **$not(x)**
Category: conditional

Description:

Returns true if x is empty.

Example:

The following statements will return the values indicated:

```
$set(foo,)
$not(%foo%)   ==>  "1"   (False)

$not(x)       ==>  ""    (True)
$not( )       ==>  ""    (True)
$not()        ==>  Error
```

7.5.21 $or

Usage: **$or(x,y,*args)**
Category: conditional

Description:

Returns true if either x or y is not empty. Can be used with an arbitrary number of arguments. The result is true if **ANY** of the arguments is not empty.

Example:

The following statements will return the values indicated:

```
$or(,)          ==>  ""    (False)
$or(,1)         ==>  "1"   (True)
$or(,A)         ==>  "1"   (True)
$or(,$gt(4,5))  ==>  ""    (False)
$or(,$lt(4,5))  ==>  "1"   (True)
$or(,,)         ==>  ""    (False)
$or(,,0)        ==>  "1"   (True)
$or(,, )        ==>  "1"   (True)
```

7.5.22 $startswith

Usage: **$startswith(text,prefix)**
Category: conditional
Implemented: Picard 1.4

Description:

Returns true if `text` starts with `prefix`. Note that the comparison is case-sensitive.

Example:

The statements below return the values indicated:

```
$startswith(The time is now.,The time)  ==>  "1"  (True)
$startswith(The time is now.,The TIME)  ==>  ""   (False)
$startswith(The time is now.,)          ==>  "1"  (True)
$startswith(,The)                       ==>  ""   (False)
$startswith(,)                          ==>  "1"  (True)
```

7.6 Information Functions

These functions provide additional system or data information. The information scripting functions are:

7.6.1 $datetime

Usage: **$datetime([format])**
Category: information
Implemented: Picard 2.3

Description:

Returns the current date and time in the specified format, which is based on the standard Python `strftime` format codes[79]. If no format is specified the date and time will be returned in the form '2020-02-05 14:26:32'. Note that any special characters such as '%', '$', '(', ')' and '' will need to be escaped as shown in the examples below.

> **Warning:** Platform-specific formatting codes should be avoided to help ensure the portability of scripts across the different platforms. These codes include: remove zero-padding (e.g.: `%-d` and `%-m` on Linux or macOS, and their equivalent `%#d` and `%#m` on Windows); element length specifiers (e.g.: `%3Y`); and hanging '%' at the end of the format string.

---

[79] https://strftime.org

Example:

The following statements will return the values indicated:

```
$datetime()                         ==>  "2020-02-05 14:26:32"
$datetime(\%Y-\%m-\%d \%H:\%M:\%S)  ==>  "2020-02-05 14:26:32"
$datetime(\%Y-\%m-\%d)              ==>  "2020-02-05"
$datetime(\%H:\%M:\%S)              ==>  "14:26:32"
$datetime(\%B \%d, \%Y)             ==>  "February 05, 2020"
```

7.6.2 $matchedtracks

Usage: **$matchedtracks()**
Category: information
Implemented: Picard 0.12

Description:

Returns the number of matched tracks within a release. Note that this only works in File Naming scripts.

Example:

The following statements will return the values indicated:

```
$matchedtracks()  ==>  "3" (if three of the tracks were matched)
```

7.7 Loop Functions

These functions provide the ability to repeat actions based on the contents of a multi-value variable or the result of a conditional test. The loop scripting functions are:

7.7.1 $foreach

Usage: **$foreach(name,code,separator="; ")**
Category: loop
Implemented: Picard 2.3

Description:

Iterates over each element found in the multi-value variable `name`, executing `code` during each iteration. Before each iteration, the element value is first stored in the variable `_loop_value` and the count is stored in the variable `_loop_count`. This allows the element or count value to be accessed within the code script.

A literal value representing a multi-value can be substituted for `name`, using the `separator` (or a semicolon followed by a space "; " if not passed) to coerce the value into a proper multi-valued variable.

Example:

The following statements will perform the processing indicated:

```
$noop( Mark all listed tags for deletion from the files. )
$foreach(genre; comment; year,$delete(%_loop_value%))

$noop( Create an 'artist_count' tag with a count of all artists
      listed for the track. )
$foreach(%artists%,$set(artist_count,%_loop_count%))

$noop( Create a separate tag for each artist listed for the
      track as 'artist_1', 'artist_2', etc. )
$foreach(%artists%,$set(artist_%_loop_count%,%_loop_value%))
```

7.7.2 $while

Usage: **$while(condition,code)**

Category: loop

Implemented: Picard 2.3

Description:

Executes `code` repeatedly until `condition` no longer evaluates to True. For each loop, the count is stored in the variable `_loop_count`. This allows the count value to be accessed within the `code` script.

Note: The function limits the maximum number of iterations to 1000 as a safeguard against accidentally creating an infinite loop.

Example:

The following statement will set `return` to "Echo… echo… echo…":

```
$set(return,Echo...)$while($lt(%_loop_count%,2),$set(return,%return% echo...))
```

7.8 Miscellaneous Functions

The miscellaneous scripting functions are:

7.8.1 $noop

Usage: **$noop(…)**
Category: miscellaneous

Description:

Does nothing and always returns an empty string. This is useful for comments or disabling a block of code.

Example:

The following statements will return the values indicated:

```
$noop( A comment. )         ==>  ""
$noop($set(foo,Testing...)) ==>  "" (and "foo" is not set)
```

USING PICARD

There are four stages to using Picard to process your audio files:

8.1 Retrieving Album information

This stage identifies the album on MusicBrainz that will provide the information used for tagging the files, and retrieves the metadata from the MusicBrainz database. There are a few different methods available, depending on the information currently available on your system (e.g.: metadata exising in the files, or having the source CD available).

There are basically four main methods used to retrieve album information from the MusicBrainz database.

8.1.1 Lookup CD

This is the preferred method of automatically identifying the album to retrieve, and should be used when you have the CD available. Typically this would be used right after ripping the audio files from the CD. When initiated, the table of contents (toc) is read from the CD and a request is sent to MusicBrainz to return a list of the releases that match the toc. If there are any matches, then they will be listed for you to select the one to use. If there are no matches or none of the matches are correct, you can search the database manually for the matching album, and are given the option of attaching the toc from your CD to the selected release for future lookup.

The steps to follow to lookup a CD are:

1. Make sure the CD is inserted in the drive, and select *"Tools → Lookup CD. . . → (drive to use)"*. The CD toc will be calculated and sent to MusicBrainz, and a list of matching releases will be displayed.
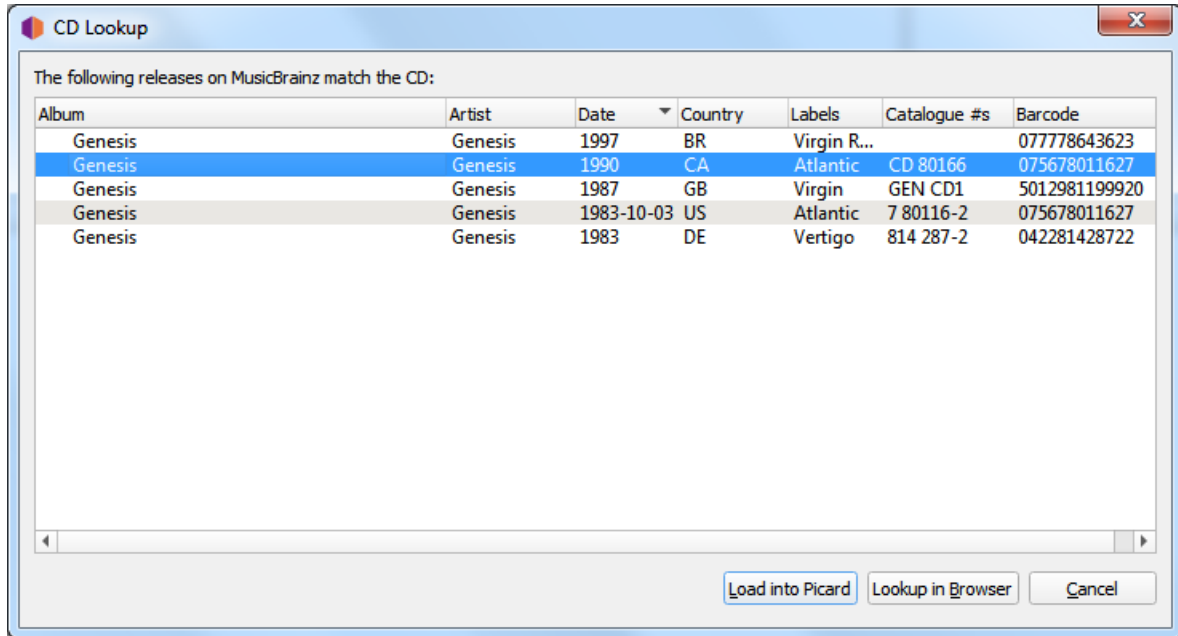
2. Select the correct release from the list. This will load the information for the release into Picard.

A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned to the track.
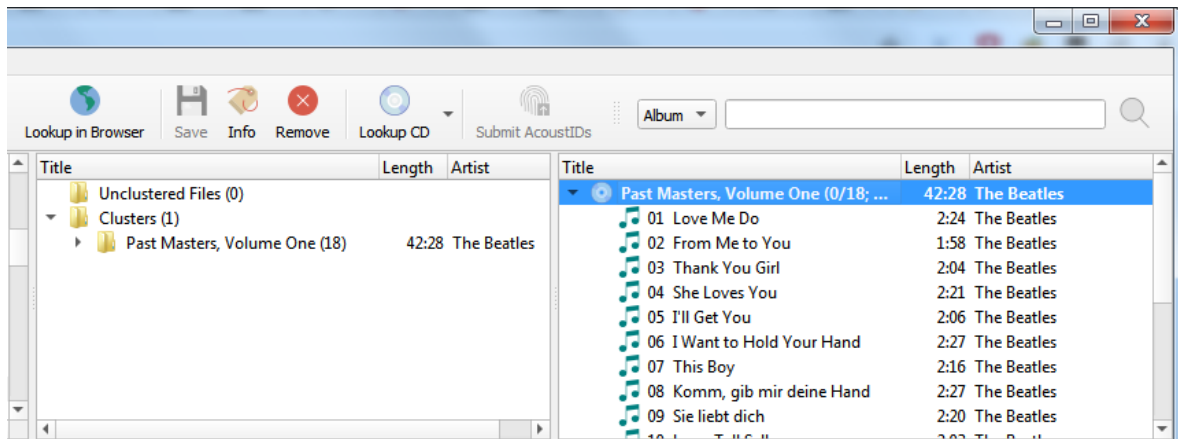
3. If there are no matches or none of the matches are correct, use the *"Lookup in Browser"* option to locate the correct release. Enter the release title or artist and initiate the search. You will be provided with a list of the releases that match your search criterion and have the same number of tracks as your

CD toc.

4. Use the green arrow to load the information for a release into Picard. In addition, you can select the release and attach the toc.

5. If none of the releases displayed are correct, you have the option to add a new release (with some information automatically included).

8.1.2 Lookup Files

If you don't have the CD available, and your files are grouped by album, this is the preferred method of automatically identifying the album to retrieve. This is done by grouping the files into album clusters in Picard and then perform the lookup. Picard will try to match the entire set of clustered files to the same release.

The steps to follow to lookup files are:
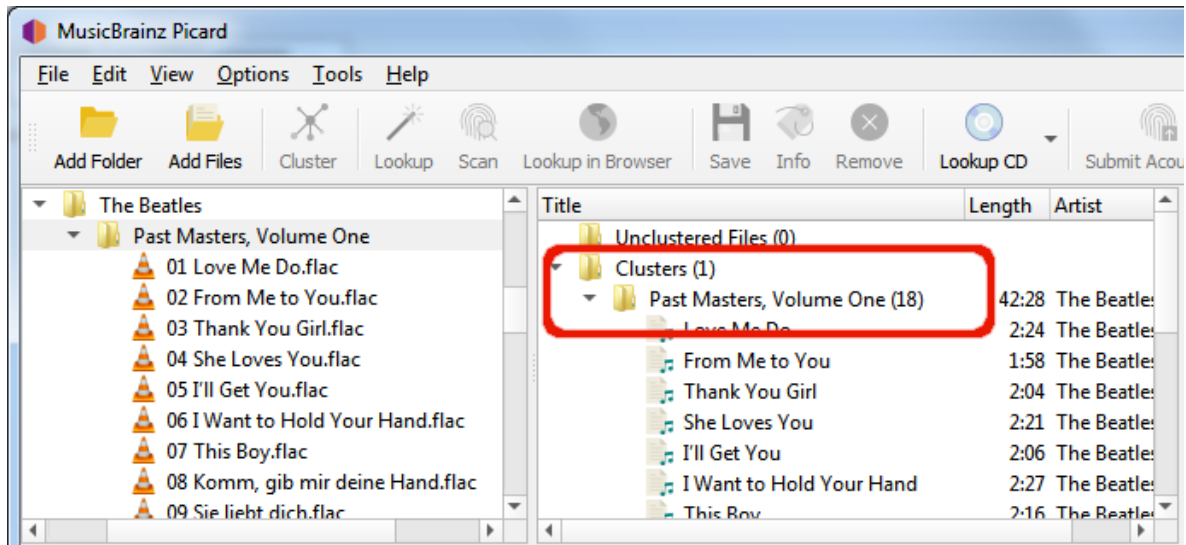
1. Add your files using *"Files → Add Files..."* or *"Files → Add Folder..."*. For ease of use it is recommended to use the internal File Browser to manage file system interactions. This is enabled from *"View → File Browser"*.

2. Drag the selected directory or files to the "Unclustered Files" folder, and wait for Picard to process the files - the names will turn from grey to black.

3. Use *"Tools → Cluster"* to group the files into album clusters.

4. Select a clustered album and use *"Tools → Lookup"* to lookup the cluster. Depending on your previous metadata, the album will show up in the right-hand pane.

   A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned to the track.

If no album was retrieved, or if the album retrieved was incorrect, you may have to try a different method such as scanning the files or a manual lookup.

8.1.3 Scan Files

If your files are not grouped into albums and you don't have the CD available, this is the only remaining method of automatically identifying the album to retrieve. This is done by scanning the files to obtain their AcoustID fingerprints and then perform the lookup for the individual files by fingerprint. The album(s) matching the files will show up in the right-hand pane based on a "best match" using the Preferred Releases settings in the Metadata options.

The steps to follow to lookup files are:

1. Add your files using *"Files → Add Files. . . "* or *"Files → Add Folder. . . "*. For ease of use it is recommended to use the internal File Browser to manage file system interactions. This is enabled from *"View → File Browser"*.

2. Drag the selected directory or files to the "Unclustered Files" folder, and wait for Picard to process the files - the names will turn from grey to black.
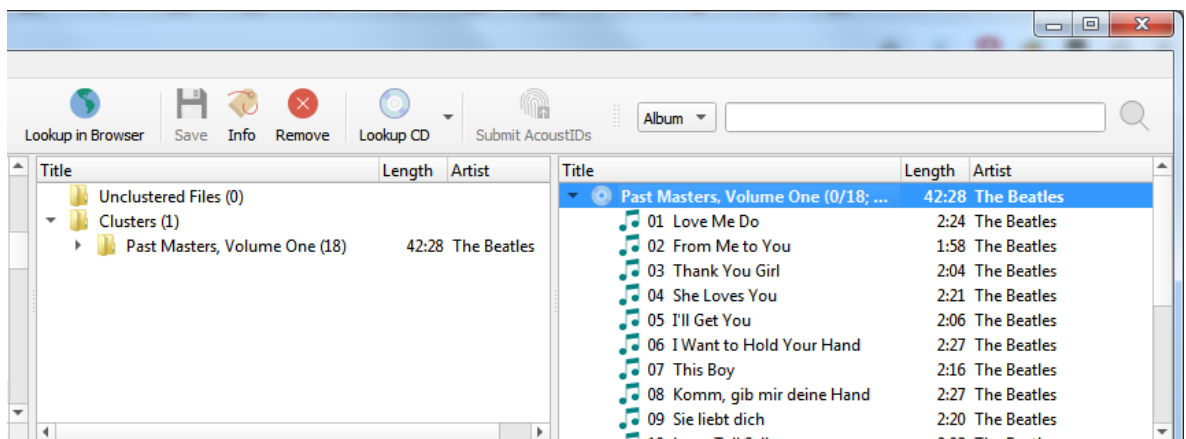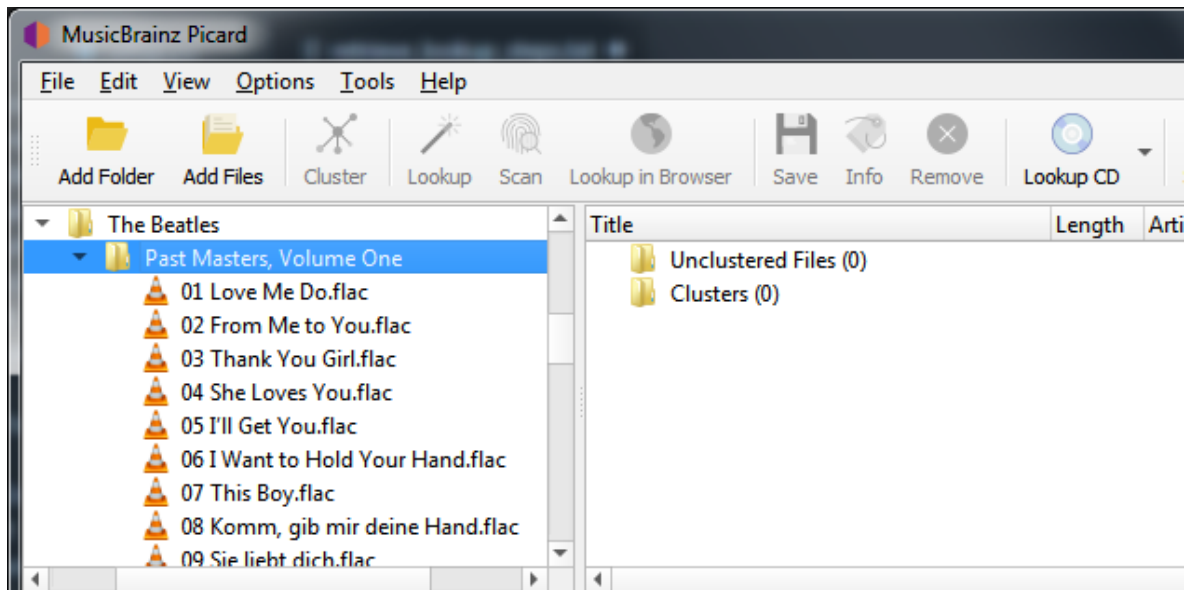

3. Use *"Tools → Cluster"* to group the files into album clusters.

4. Select a clustered album and use *"Tools → Lookup"* to lookup the cluster. Depending on your previous metadata, the album will show up in the right-hand pane.

   A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned to the track.

   If no album was retrieved, or if the album retrieved was incorrect, you may have to try a different method such as scanning the files or a manual lookup.

8.1.4 Lookup in Browser

If none of the automated methods are available, or don't produce the desired results, you have the option of retrieving the album information by having Picard initiate a search on the MusicBrainz website using your web browser. There are two methods of initiating this search. The first method searches based on the tag information from the selected files.

The steps to follow to manually lookup an album on MusicBrainz are:

1. Add your files using *"Files → Add Files..."* or *"Files → Add Folder..."*. For ease of use it is recommended to use the internal File Browser to manage file system interactions. This is enabled from *"View → File Browser"*.
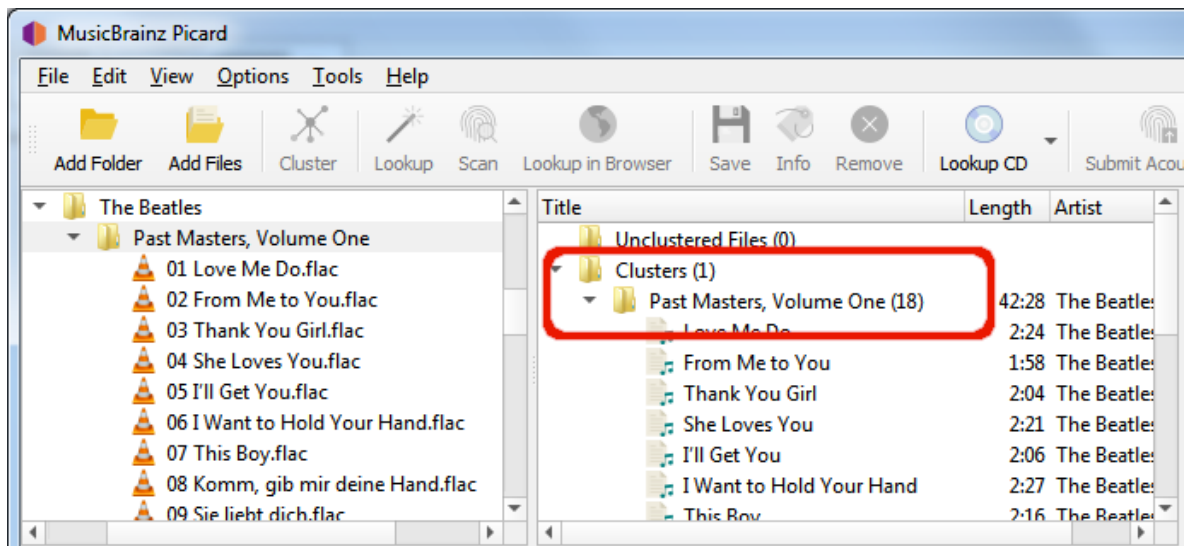

2. Drag the selected directory or files to the "Unclustered Files" folder, and wait for Picard to process the files - the names will turn from grey to black.
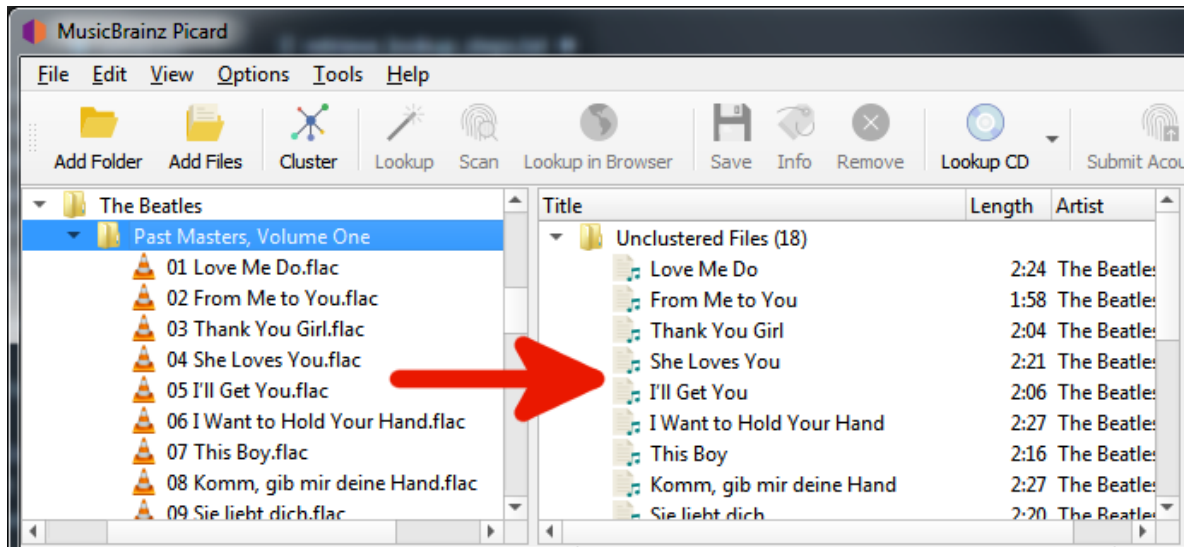
3. Use *"Tools → Cluster"* to group the files into album clusters if you want lookup a cluster.

4. Select a file or clustered album and use *"Tools → Lookup in Browser"* to initiate the search in your browser using the currently available metadata.

5. If you want to revise or refine your search criteria, make the desired changes at the bottom of the web page and click the "Search" button to re-initiate the search.

6. Use the green arrow to load the information for a release into Picard.

7. A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned to the track.

8.1.5 Manual Lookup

The second browser search method uses manually entered information as the search criterion.

The steps to follow to manually lookup an album on MusicBrainz are:

1. Add your files using *"Files → Add Files. . . "* or *"Files → Add Folder. . . "*. For ease of use it is recommended to use the internal File Browser to manage file system interactions. This is enabled from *"View → File Browser"*.
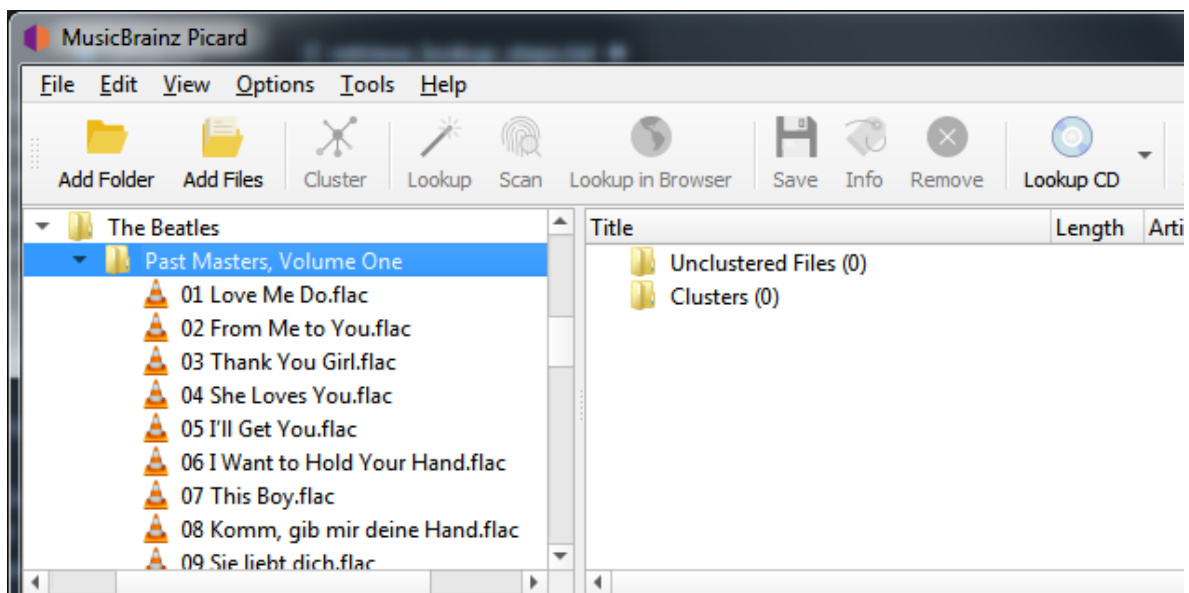
2. Drag the selected directory or files to the "Unclustered Files" folder, and wait for Picard to process the files - the names will turn from grey to black.
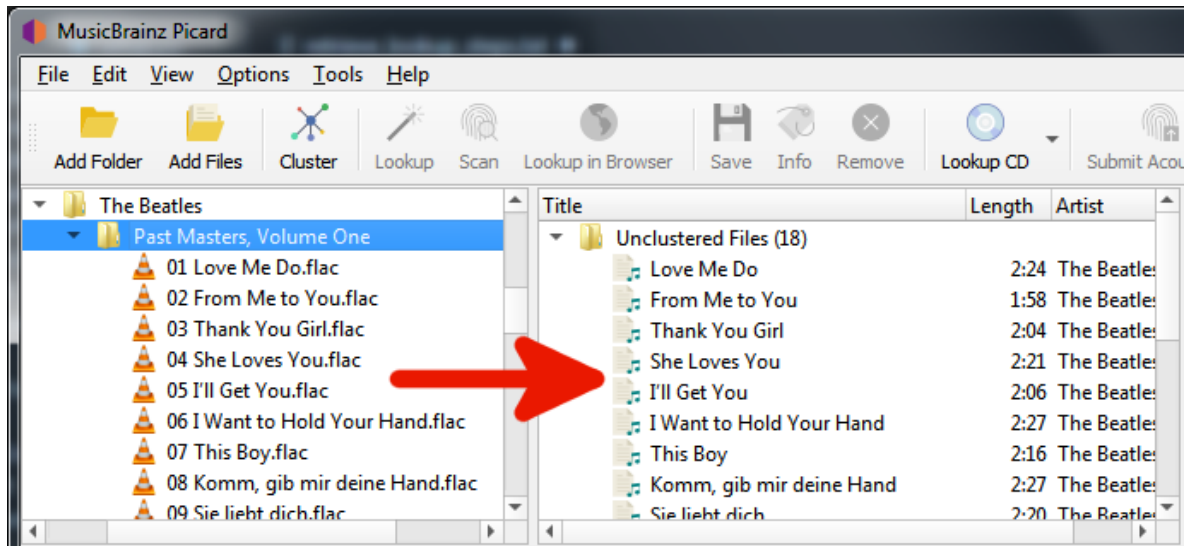
3. Enter your search information into the search box and select the type of records to search, then click the magnifying glass symbol to initiate the search. This will open the MusicBrainz website[80] in your browser.
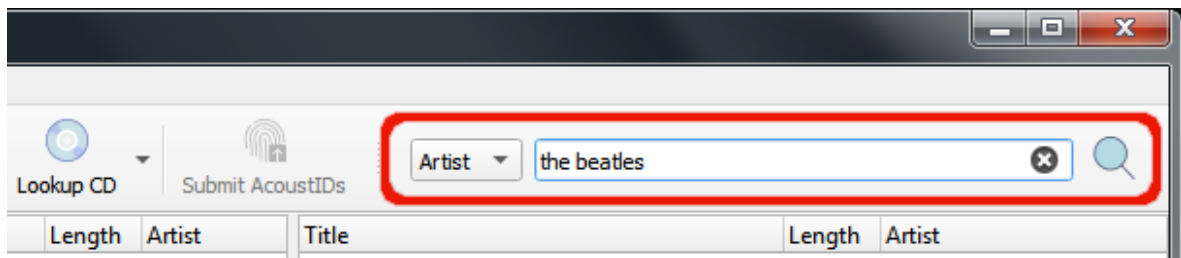
4. Continue to drill down by clicking on the appropriate links until you get to the release that you want to retrieve.

---

[80] https://musicbrainz.org
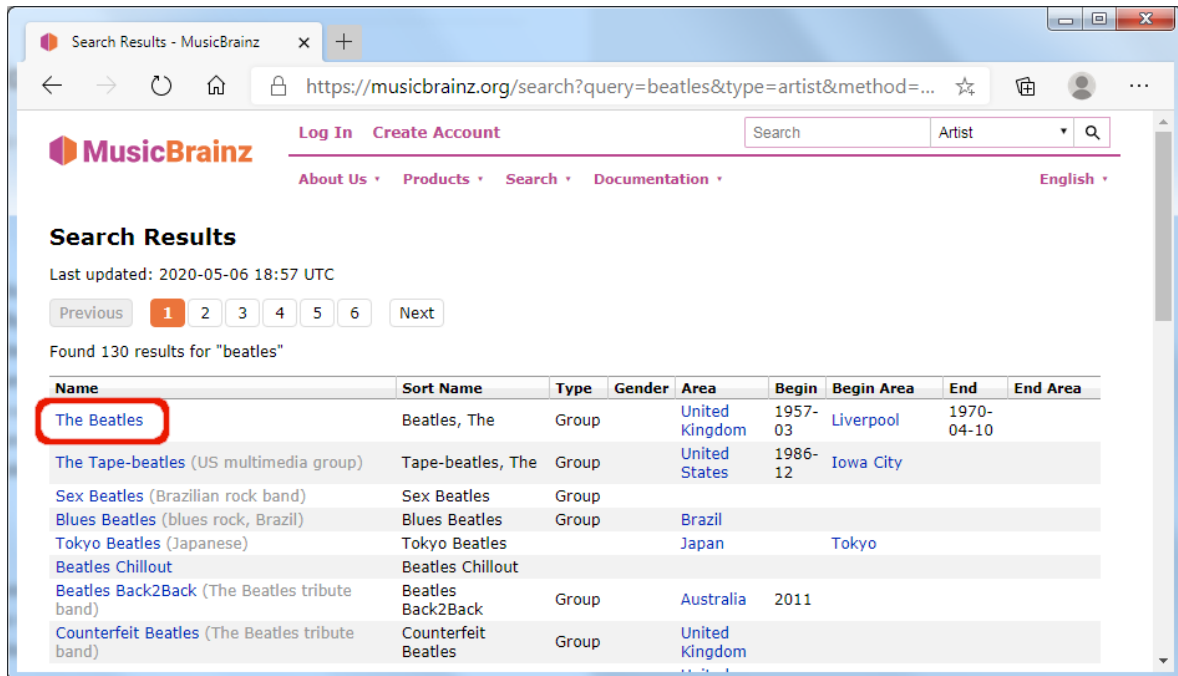
5. Use the green arrow to load the information for a release into Picard.
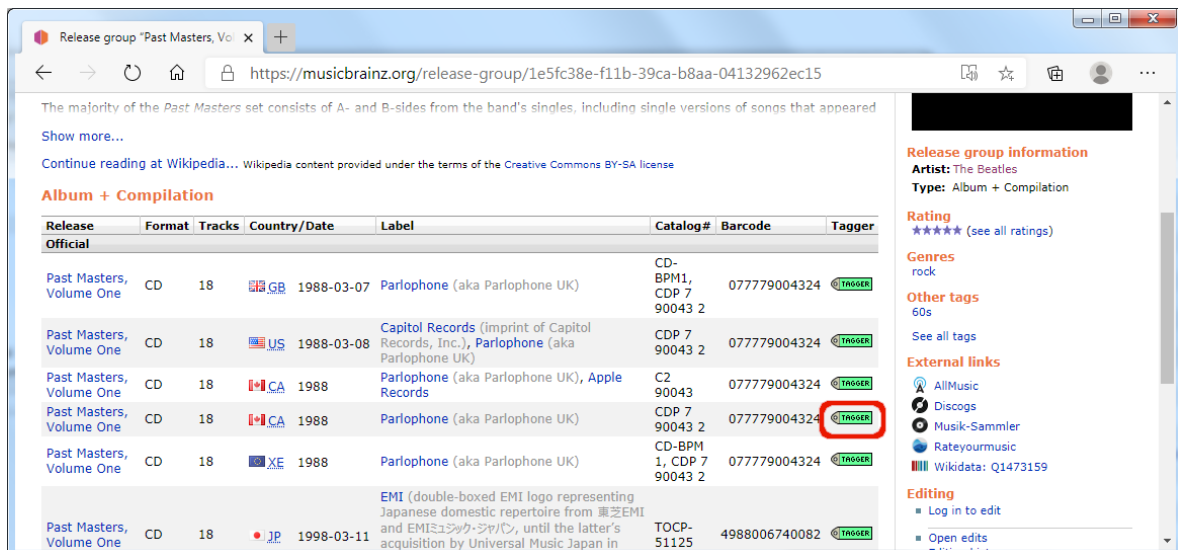
6. A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned to the track.
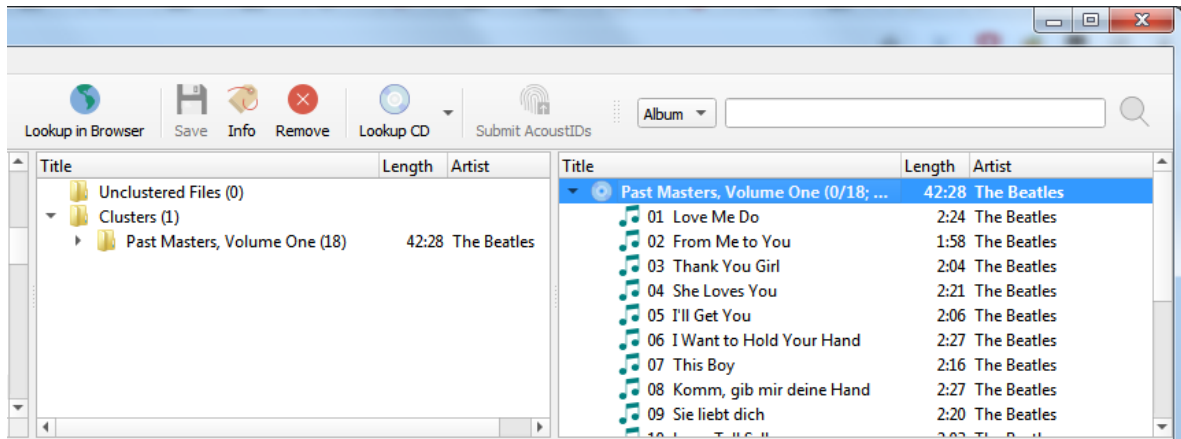
Note: If you enter a link to the desired entry (e.g.: `https://musicbrainz.org/release/9383a6f5-9607-4a36-9c68-8663aad3592b`) in the search box in Picard, the entry will be loaded directly without opening a browser window.

8.2 Matching Files to Tracks

This stage is where individual files are matched to specific tracks in the information retrieved from the MusicBrainz database.

Once you have retrieved the desired album information into the right-hand pane, the next step is to match the files from the left-hand pane to the corresponding track in the right-hand pane. A music symbol in front of a track number in the right-hand pane indicates that there has been no file assigned to the track. In some cases, Picard may have already tried to do the matching for you. If the matching wasn't done automatically, drag the appropriate files onto the appropriate album and track.
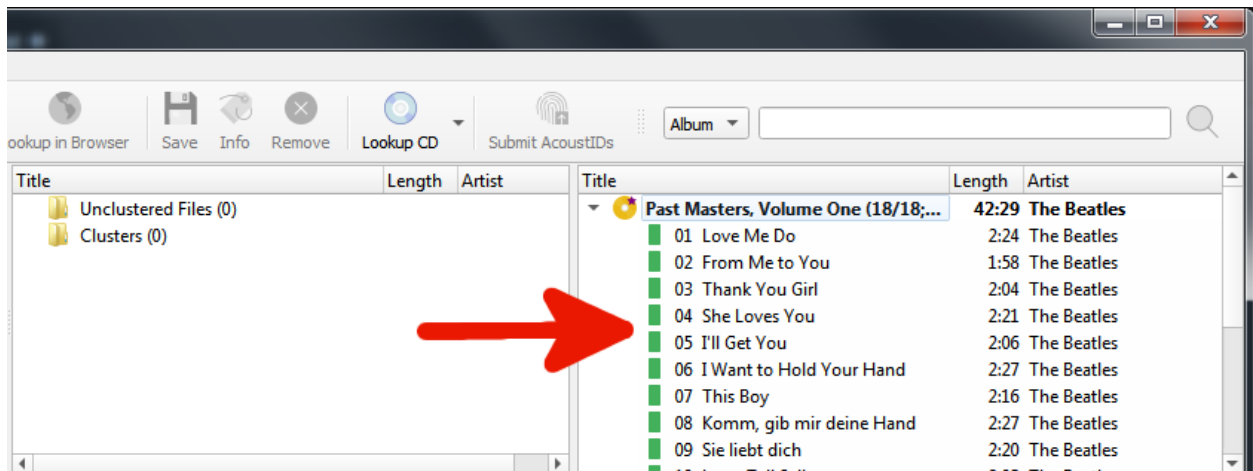
Depending on your previous metadata, Picard will try to guess the matching tracks. The order is green > yellow > orange > red, where green is the best match. If you are seeing a lot of red and orange, it could mean that Picard has guessed incorrectly, or that your files didn't have a lot of previous metadata to work with. If this is the case, it's recommended to select a track and compare the "Original Values" and "New Values" in the metadata pane. If there is an incorrect match, simply drag the track to its correct spot in the right-hand pane.
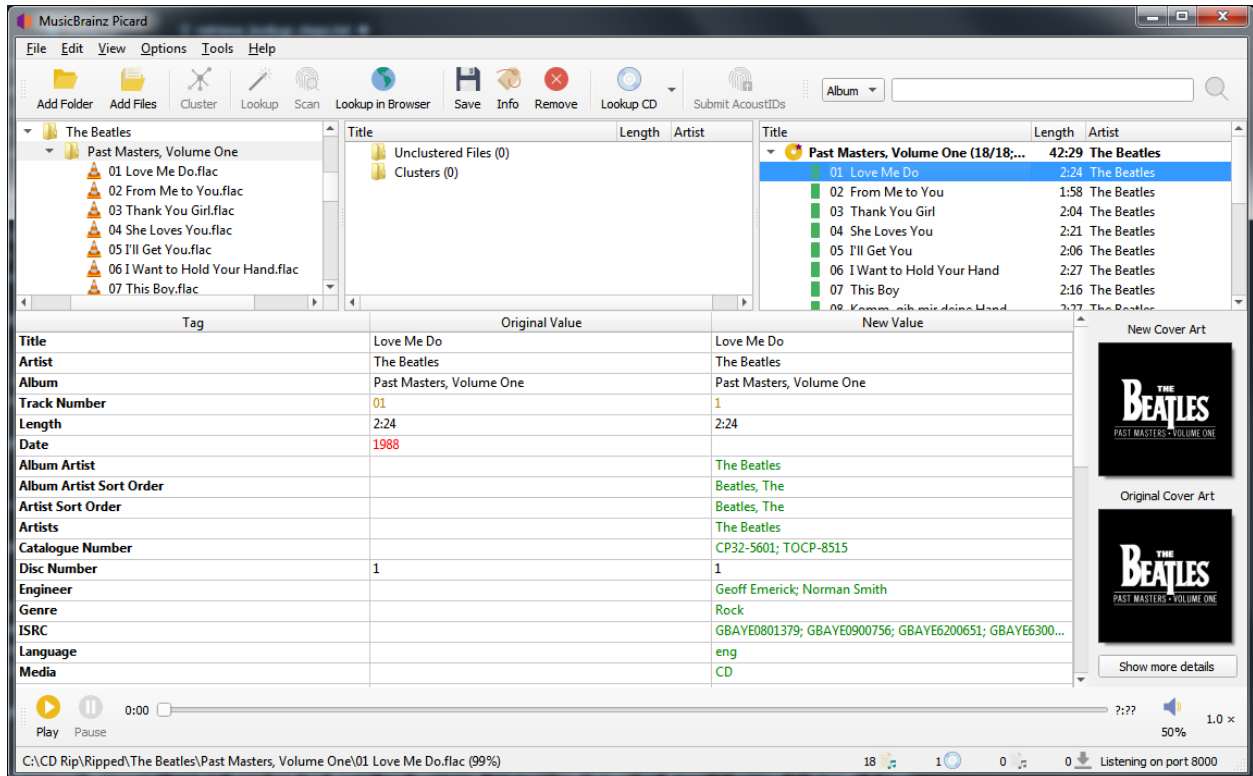
Right-clicking an item in the track list brings up a menu of commands, including "Info", "Open in Player", "Open Containing Folder", "Search for similar tracks", "Looup in Browser", "Generate AcoustID Fingerprints", "Save" and "Remove". In addition, you can re-run any associated plugins or scripts against only the selected item. Right-clicking an items in the left-hand pane will bring up a similar menu of commands.

When you select an item in the right-hand pane, the original and new metadata for the item is displayed. Right-clicking a line in the tag list brings up a menu of commands, including "Edit", "Add to 'Preserved Tags' List", "Remove" and "Add New Tag", along with an option to display the changed tags first.

8.3 Setting the Cover Art

Depending on the option settings, you can change or confirm the cover art to save with a track or album.

Once the release information has been downloaded, selecting an album or track in the right-hand pane will display both the original and new coverart for the selected item. You can easily replace the coverart image used for the selected item by dragging the image from the file browser and dropping it on the New Cover Art image.
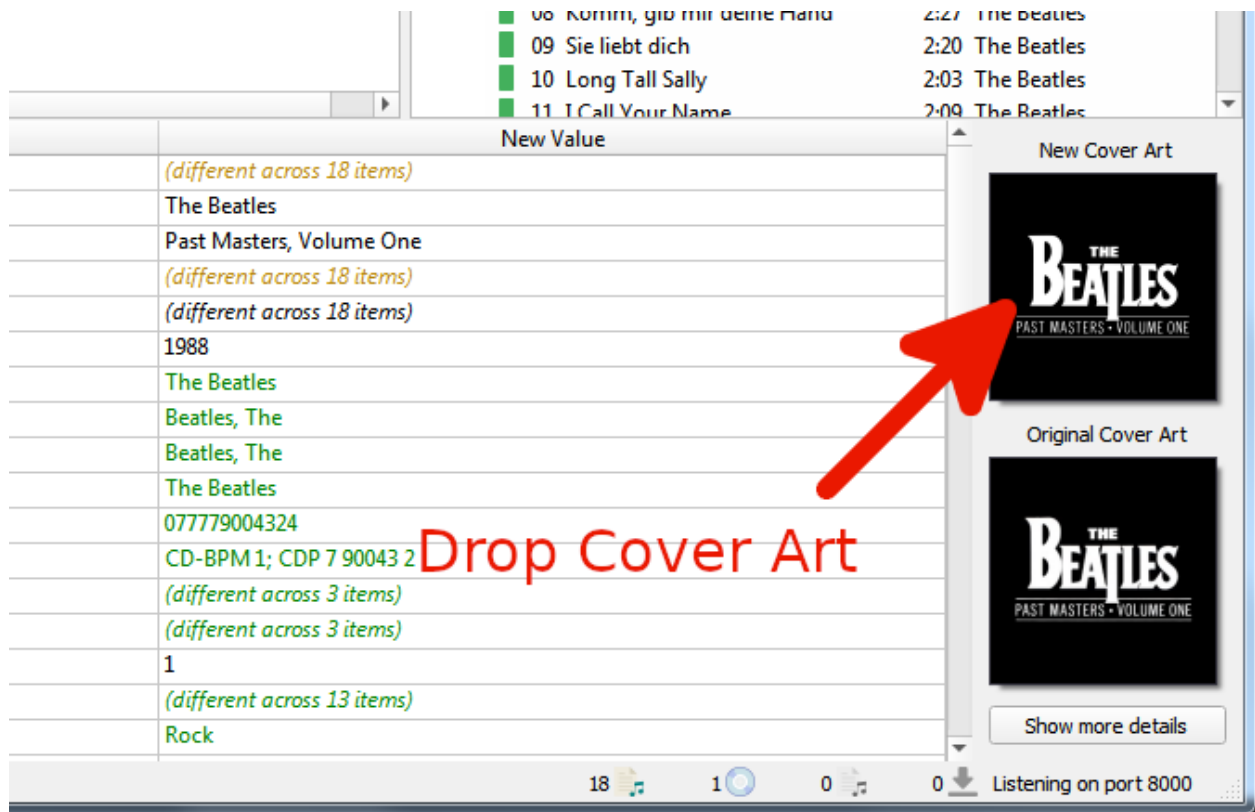
Right-clicking on the images brings up a menu of options including "Show more details", "Keep original cover art", and options for the way that images dropped onto the selection are processed. Selecting "Show more details" will bring up a new window as:

Double-clicking an image will open the image file in the system default program for the image type.

8.4 Saving Updated Files

This stage is where Picard updates the matched files with the metadata retrieved in the first stage, based on the settings configured in the Options. This may also include renaming the files and placing them in a different directory.

When you are satisfied that your files have been properly matched to tracks in the right-hand pane, select the album you want to save in the right-hand pane and use *"File → Save"* to save the files. A green check mark means the file was saved to its proper location.

Title	Length	Artist
▼ ◉ **Past Masters, Volume One (18/18;...**	**42:28**	**The Beatles**
✓ 01 Love Me Do	2:24	The Beatles
✓ 02 From Me to You	1:58	The Beatles
✓ 03 Thank You Girl	2:04	The Beatles
✓ 04 She Loves You	2:21	The Beatles
✓ 05 I'll Get You	2:06	The Beatles
✓ 06 I Want to Hold Your Hand	2:27	The Beatles
✓ 07 This Boy	2:16	The Beatles
✓ 08 Komm, gib mir deine Hand	2:27	The Beatles
✓ 09 Sie liebt dich	2:20	The Beatles

Once the files have been saved successfully, you can remove the album from the right-hand pane by selecting it and using *"Edit → Remove"*. Note that this only removes the album from Picard and does not remove the files themselves.

WORK FLOW RECOMMENDATIONS

This section provides some recommended workflows for various tagging scenarios. These workflows are based on what are believed to be best practices.

The scenarios covered include:

1. *When the CD is available*

2. *When files are grouped by album*

3. *When files are not grouped but have some metadata*

4. *When files are not grouped and have little or no existing metadata*

Note: Regardless of whether or not it's one of the the workflows listed, it is **strongly** recommended that you make a backup copy of the files being processed and initially process a copy of your music files. This will help to ensure that Picard is properly configured (e.g.: settings, scripts, and plugins) and produces the expected and desired results.

9.1 When the CD is available

This is perhaps the best case scenario, because it provides the greatest chance of tagging your music files with the most accurate match from the MusicBrainz database. It is also one of the easier methods for looking up the release.

1. Rip the CD to music files

Extract the music filed from the CD using your favorite ripping program (e.g.: Exact Audio Copy[81] for Windows or Whipper[82] for Linux). The format for the output files depends on your personal preference and the formats supported by your player. A popular format is FLAC, which is a compressed lossless format.

2. Lookup the CD on MusicBrainz

With the CD in the drive, it can be looked up automatically using the *"Tools → Lookup CD"* command. See the *Lookup CD* section for detailed instructions.

---

[81] http://exactaudiocopy.de/
[82] https://github.com/whipper-team/whipper

3. Select the correct release

If there is only one release that matches the disc id for your disc, it will be loaded automatically. Before proceeding, please check to ensure that it properly matches your CD (e.g.: release country, date and label, catalog number, barcode, media type, and cover art). This is especially important if you are going to submit any information such as disc id or AcoustID fingerprints.

4. Load the files

Drag the files or folder from the browser to the "Unclustered Files" secion in the left-hand pane. You do not need to scan or cluster them.

5. Match the files to the tracks on the release

Drag the files from the left-hand pane and drop them on the release in the right-hand pane. Check that each track on the release is associated with only one file. The release icon should turn gold. See the *Matching Files to Tracks* section for details.

6. Verify the metadata and cover art

Check that the metadata and cover art image for the release and tracks are what you want. Adjust if required. See the *Setting the Cover Art* section for details.

7. Save the files

Save the files using the *"File → Save"* command. See the *Saving Updated Files* section for details.

8. Calculate and submit AcoustID fingerprints

This step is optional, but appreciated because it will help identify the files for others to look up for tagging.

Select the album entry in the right-hand pane and calculate the AcoustID fingerprints using *"Tools → Generate AcoustID Fingerprints"*. Once the fingerprints have been calculated, submit them using *"Files → Submit AcoustIDs"*.

Note: AcoustID fingerprints should only be submitted after the files have been tagged with MusicBrainz metadata, and you have verified that the files have been matched to the correct track on the proper release.

9.2 When files are grouped by album

If the music files to be processed are already grouped into folders by album, then the process of looking up the release in the MusicBrainz database is greatly simplified because Picard works best when processing one album at a time.

1. Load the files

Drag the files or folder from the browser to the "Unclustered Files" secion in the left-hand pane.

2. Cluster and lookup the files

Select the files in the left-hand pane and combine them into an album cluster using the *Tools → Cluster"* command. Select the cluster in the left-hand pane and initiate the lookup using the *Tools → Lookup"* command. See the *Lookup Files* section for details.

3. Select the correct release

If there is only one release that matches the lookup, it will be loaded automatically. Before proceeding, please check to ensure that it properly matches your album (e.g.: release country, date and label, catalog number, barcode, media type, and cover art). This is especially important if you are going to submit any information such as AcoustID fingerprints.

4. Match the files to the tracks on the release

Drag the files from the left-hand pane and drop them on the release in the right-hand pane. Check that each track on the release is associated with only one file. The release icon should turn gold. See the *Matching Files to Tracks* section for details.

5. Verify the metadata and cover art

Check that the metadata and cover art image for the release and tracks are what you want. Adjust if required. See the *Setting the Cover Art* section for details.

6. Save the files

Save the files using the *"File → Save"* command. See the *Saving Updated Files* section for details.

7. Calculate and submit AcoustID fingerprints

This step is optional, but appreciated because it will help identify the files for others to look up for tagging.

Select the album entry in the right-hand pane and calculate the AcoustID fingerprints using *"Tools → Generate AcoustID Fingerprints"*. Once the fingerprints have been calculated, submit them using *"Files → Submit AcoustIDs"*.

Note: AcoustID fingerprints should only be submitted after the files have been tagged with MusicBrainz metadata, and you have verified that the files have been matched to the correct track on the proper release.

9.3 When files are not grouped but have some metadata

In this situation, you will need to feed batches of files to Picard to process. In order to minimize the performance impact, it is recommended to keep the batches relatively small (i.e.: approximately 200 files at most in a single batch). Picard will try to group them into clusters based on the metadata currently existing in the files.

Note: This workflow will likely only partially match the files to a release in each batch processed. This

---

means that an album may not be fully matched, tagged and renamed until multiple batches have been pro-
cessed.

1. Load the files

> Drag the batch of files to process from the browser to the "Unclustered Files" secion in the
> left-hand pane.

2. Cluster and lookup the files

> Select the files in the left-hand pane and combine them into album clusters using the *Tools →*
> *Cluster"* command. Picard will attempt to cluster the files based on their existing metadata.
> Select the desired cluster(s) in the left-hand pane and initiate the lookup using the *Tools →*
> *Lookup"* command. See the *Lookup Files* section for details.

3. Match the files to the tracks on the release

> Drag the files from the left-hand pane and drop them on the release in the right-hand pane.
> Check that each track on the release is associated with only one file. The release icon will likely
> remain silver, indicating that not all tracks have been matched to files. See the *Matching Files*
> *to Tracks* section for details.

4. Verify the metadata and cover art

> Check that the metadata and cover art image for the release and tracks are what you want.
> Adjust if required. See the *Setting the Cover Art* section for details.

5. Save the files

> Save the files using the *"File → Save"* command. See the *Saving Updated Files* section for
> details.

Note: It is not recommended to submit AcoustID fingerprints for files matched in this way, because it is
virtually impossible to verify that your files actually match the recordings being matched.

9.4 When files are not grouped and have little or no existing metadata

This is perhaps the worst case scenario, because it provides the greatest chance of tagging your music files
with an incorrect match from the MusicBrainz database.

In this situation, you will need to feed batches of files to Picard to process. In order to minimize the
performance impact, it is recommended to keep the batches relatively small (i.e.: approximately 200 files at
most in a single batch). Picard will try to group them into clusters based on their AcoustID fingerprints.

Note: This workflow will likely only partially match the files to a release in each batch processed. This
means that an album may not be fully matched, tagged and renamed until multiple batches have been pro-
cessed.

---

1. Load the files

Drag the batch of files to process from the browser to the "Unclustered Files" secion in the left-hand pane.

2. Scan the files

Select the files in the left-hand pane and scan them using the *Tools → Scan"* command. Picard will attempt to calculate the AcoustID fingerprint for each of the files and then retrieve releases with matching recordings. See the *Scan Files* section for details.

3. Match the files to the tracks on the release

Drag the files from the left-hand pane and drop them on the release in the right-hand pane. Check that each track on the release is associated with only one file. The release icon will likely remain silver, indicating that not all tracks have been matched to files. See the *Matching Files to Tracks* section for details.

4. Verify the metadata and cover art

Check that the metadata and cover art image for the release and tracks are what you want. Adjust if required. See the *Setting the Cover Art* section for details.

5. Save the files

Save the files using the *"File → Save"* command. See the *Saving Updated Files* section for details.

EXTENDING PICARD

There are two primary ways that the functionality of MusicBrainz Picard can be extended: *plugins* and *scripts*.

Plugins can be installed / uninstalled and enabled / disabled from the Options menu. Installed plugins are loaded during the startup of Picard, and are made available to the program.

Scripts are stored within the user settings, and are managed from the *"Options → Options…"* menu.

10.1 Plugins

Plugins are written in Python, and are registered to the appropriate hooks. Each plugin has its own version identifier, but also lists the plugin API versions that it supports. When loading a plugin, Picard first compares its list of API versions to the plugin's supported versions to ensure that the plugin will operate correctly. The Picard API versions indicate the version of the program in which the plugin API was last updated and any plugin APIs with which it is backwards compatible.

Hooks are connections to the various objects in Picard that call a specific type of plugin. During the normal running of Picard, when it encounters a hook it will first retrieve a list of all plugins registered for that specific hook, and then execute them sequentially in order based upon the priority specified when the plugin was registered to the hook.

There are a few different types of plugins, including:

Metadata processors: These plugins can access and modify the metadata when it is loaded from MusicBrainz. They are registered with `register_album_metadata_processor()` or `register_track_metadata_processor()`. These are what you might call "automatic" because they operate without any user intervention. An example is the Classical Extras plugin.

Cover art providers: These plugins provide another cover art source, and are registered with `register_cover_art_provider()`. They are also "automatic" in that they load album art without user intervention, although they must be enabled by the user in the Cover Art options. The Fanart.tv plugin is an example.

Scripting function: Some plugins just provide additional scripting functions for use in *"Options → Scripting"* or the renaming script. These are registered with `register_script_function()`. Keep tag, which provides the `$keep()` function, is an example.

Context menu actions: Plugins can register actions that can be activated manually via the context menu. This is what the Load as non-album track plugin does. Another example is Generate Cuesheet. These are registered with `register_album_action()`, `register_track_action()`, `register_file_action()`, `register_cluster_action()` or `register_clusterlist_action()`.

File formats: Plugins can also provide support for new file formats not yet supported by Picard. These are registered with `register_format()`.

Event processors: Plugins can execute automatically based on certain event triggers. These are registered with `file_post_load_processor()`, `file_post_save_processor()`, `file_post_addition_to_track_processor()`, `file_post_removal_from_track_processor()` or `album_post_removal_processor()`.

Note that plugins are not limited to one of those areas. A single plugin could implement all of the above, but most existing plugins focus on one.

The *Plugins API* provides information regarding the different plugin hooks available, along with some examples of their use. There is also a list of the available plugins[83] that have been submitted to the MusicBrainz Picard repository shown on the Picard website.

10.2 Scripts

There are two types of scripts used in Picard: the file naming script and tagging scripts. These are managed from the "File Naming" and "Scripting" sections of the *"Options → Options..."* menu. All scripts are written using the *Picard scripting language*. Scripts are often discussed in the MetaBrainz Community Forum[84], and there is a thread specific to file naming and script snippets[85].

10.2.1 File Naming Script

There is only one file naming script defined in a user's settings, although it can vary from a simple one-line script such as `%album%/%title%` to a very complex script using different file naming formats based on different criteria. In all cases, the files will be saved using the text output by the script.

Note: Any new tags set or tags modified by the file naming script will not be written to the output files' metadata.

---

[83] https://picard.musicbrainz.org/plugins/
[84] https://community.metabrainz.org/
[85] https://community.metabrainz.org/t/repository-for-neat-file-name-string-patterns-and-tagger-script-snippets/2786/

10.2.2 Tagging Scripts

There can be multiple tagging scripts defined in a user's settings. Individual scripts can be enabled or disabled, and the order of execution of the scripts can be set. Whenever a script is run automatically (i.e.: when an album is loaded), it is processed once for each track in the album that triggered the run. For example, if there are two tagging scripts enabled (A and B) and an album with three tracks is loaded, the scripts will be processed in the following order:

1. Script A Track 1;

2. Script A Track 2;

3. Script A Track 3;

4. Script B Track 1;

5. Script B Track 2;

6. Script B Track 3.

Metadata updates are not shared between tracks, so you cannot append data from one track to a tag in another track.

Any new tags set or tags modified by the tagging scripts will be written to the output files' metadata, unless the tag name begins with an underscore. These "hidden" tags are typically used as variables to hold temporary values that are used later in either the tagging or file naming scripts. Tagging scripts are run once for each track in the data, using the metadata for that track.

Tagging scripts can also be run manually by right-clicking either an album or a track in the right-hand pane in Picard. If run from the album entry, the script is run for each track in the album. If run from an individual track, the script is only run for that track.

10.3 Processing Order

In order to make effective use of plugins and scripts, it is important to understand when each is processed in relation to the others. As a general statement, plugins are always processed before scripts. Plugins of the same type will be executed in order based upon the priority specified when the plugin was registered.

10.3.1 Startup

During program startup, plugins with the following hooks are processed, and any additional functionality that they provide will be available immediately:

- File Formats

- Cover Art Providers

- Tagger Script Functions

- Context Menu Actions

- Option Pages

10.3.2 Loading a Release

When data gets loaded from MusicBrainz (while the album shows the "loading" status in the right-hand pane), the following are processed:

- Metadata Processor Plugins
- Tagging Scripts

Plugins have access to the raw data loaded from MusicBrainz and are processed before scripts, in the order of priority set when the plugin was registered.

Scripts are processed in the order set in the Options menu.

Note: Tagging scripts are always run against metadata loaded from MusicBrainz, and exactly after the data gets loaded and before files get matched. They are one of the last steps in the loading process. Tagging scripts do not have access to metadata stored in existing files.

10.3.3 Loading Music Files

After a file has been loaded into Picard, plugins registered with `file_post_load_processor()` are executed. This could, for example, be used to load additional data for a file.

10.3.4 Adding / Removing Files

After a file has been added to a track (on the right-hand pane of Picard), plugins registered with `file_post_addition_to_track_processor()` are executed.

After a file has been removed from a track (on the right-hand pane of Picard), plugins registered with `file_post_removal_from_track_processor()` are executed.

10.3.5 Saving Files

When files are saved, for each file the File Naming Script is first executed to determine the destination path and file name. Note that this script has no effect on the tag values written to the output file.

After a file has been saved, plugins registered with `file_post_save_processor()` are executed. This can, for example, be used to run additional post-processing on the file or write extra data. Note that the file's metadata is already the newly saved metadata.

10.3.6 Removing Albums

After an album has been removed from Picard, plugins registered with `album_post_removal_processor()` are executed.

10.3.7 Context Menus

Individual tagging scripts can be executed on-demand from the context menu displayed when right-clicking on a file, album, track, cluster or cluster list.

TROUBLESHOOTING

11.1 General Troubleshooting

11.1.1 Getting Help

If you have problems using Picard, please first check the following resources:

- For general usage information see the *Using Picard* documentation and the illustrated quick start guide[86].

- Read the *FAQ section* for common questions and problems.

- Consult the community forums[87].

- Check the download page[88] for a newer version of Picard which might solve your problem.

- If the problem is to do with a plugin, check the Picard Plugins[89] for updated plugin versions.

11.1.2 Reporting a Bug

If you think you have found a bug please check whether you are using the latest version of Picard and whether the bug has already been reported in the bug tracker[90]. If you're not sure or don't want to look through the existing tickets, ask on the community forums first.

If you're still convinced you have found a new bug, open a new ticket[91] providing the following information:

- Which version of Picard do you use? ("Affects Version" in the form)

- Which operating system do you use? ("Environment" in the form)

- What did you do when the bug occurred?

- What actually happened and what did you expect to happen?

- If you're using plugins, which plugins do you have enabled?

---

[86] https://picard.musicbrainz.org/docs/guide/
[87] https://community.metabrainz.org/c/picard
[88] https://picard.musicbrainz.org/downloads/
[89] https://picard.musicbrainz.org/plugins/
[90] https://tickets.musicbrainz.org/browse/PICARD
[91] https://tickets.musicbrainz.org/secure/CreateIssue.jspa?pid=10042&issuetype=1

11.1.3 Getting Logs

For many bugs, it helps developers to have a log from Picard. You can see the log by going to *"Help → View Log"*. You can also get a full debug log (better because it contains more detailed information) by starting Picard with *-d* as a command-line argument. If you're using Windows, you can change your shortcut's Target (*right click shortcut → Properties*) to:

```
"C:\Program Files\MusicBrainz Picard\picard.exe" -d
```

Pasting this log into your forum post or bug ticket can help developers and other users to resolve your issue more quickly.

> **Warning:** Please remember to first remove any personal and confidential information like user id, passwords or authorization tokens before posting or submitting any log output.

11.2 There is no coverart

There are two different problems that often fall under this topic:

11.2.1 Picard isn't finding and downloading any cover art

No cover art providers have been enabled in the configuration settings

Confirm that the *"Options → Options. . . → Cover Art"* settings have at least one cover art provider enabled. Please see the *Cover Art Providers* section for more information.

There is no cover art available from the selected providers

It's possible that the selected release does not have any cover art available from the enabled cover art providers. If a cover art image is displayed for the release on the MusicBrainz website, it is possible that the image for the release group is being displayed, or it is being provided through a third-party provider agreement. Sometimes this can be addressed by enabling the "CAA Release Group" and "Whitelist" provider options.

The selected provider is not currently available

On occasion, the server providing the cover art (e.g.: archive.org) is not available, or mirror servers have not yet been synchronized with the latest updates. In this case, you may have to wait for a few minutes before retrying your request. Reviewing the details in Picard's log often provides some insight into whether or not this is the issue.

The cover art is still a pending edit

If the cover art was recently added, the edit adding the image may not have been accepted and applied yet. You can have Picard use the cover art from pending edits by disabling the "Download only approved images" option in the Cover Art Archives subsection of the *"Options → Options. . . → Cover Art"* settings. Please see the *Cover Art Archive* section for more information.

11.2.2 Coverart that is saved with the files isn't displayed

Player doesn't support embedded cover art

Check to confirm that your player supports embedded cover art images. That support is not universal among all players. Some players support embedded images, some support images stored as files in the directory (e.g.: "cover.jpg" or "folder.jpg"), and some support both. Picard allows you to specify how the cover art images should be saved. Please see the *Location* section of the Coverart options for details.

You should also confirm that your player supports the version of the tags being written. Please see the *Tag Compatibility* section for more information.

Embedded cover image too large

Even if your cover art image has been properly embedded in the file, it's possible that your player has trouble dealing with embedded images over a certain size. If all else fails, you might try using an image with a smaller file size.

11.3 Tags are not updated or saved

There are typically four reasons that tags may not be written or updated when files are saved:

Saving tags has not been enabled in the configuration settings

Confirm that the *"Options → Save tags"* setting has been enabled. See *Action Options* for more information.

Tags are being set in the file naming script

Tags created or updated in the file naming script will not be written to the output files. This script is only used for developing the file name and directory structure for the output. If you want to set or update a tag value in a script, it must be in a tagging script. Please see the *Scripts* section for more information about the different types of scripts.

The tags begin with an underscore

Tags whose names begin with an underscore, regardless of how they are created, will not be written to the output files. These are considered variables for use within Picard rather than tags. Please see the *Tags & Variables* section for more information regarding the difference between tags and variables.

The file type does not support writing tags

Confirm that the file type that you are writing actually supports the tags that are to be written. For example, WAV files cannot be tagged due to the lack of a standard for doing so. Please see the *Tag Compatibility* section for additional information.

11.4 Files are not being saved

There are two typical scenarios where files are not being saved:

After selecting files in the right-hand pane you see a red stop like icon

This indicates an error occurred during saving. In the majority of times people see this it is because the files they want to save are write protected (either have the readonly flag set or have wrong permissions). Check that the files are not write protected and that you have the appropriate permissions before trying again.

Permission problems seem to be more common when Picard has been installed using Flatpak, or when the files are being read from or written to a samba share on the network.

Another possibility is that the total length of the destination path and file name exceeds the maximum length allowed by the operating system. If you have an extremely long path and file name, try shortening it to see it it allows the file to be saved.

In the right-hand pane you see just a musical note icon in front of the tracks

That means that this is just the track data from MusicBrainz, but no file has been associated with it. In that case the save button is disabled. Check to make sure that the files are properly matched to the tracks before trying to save again. Please see the *Matching Files to Tracks* and *Saving Updated Files* sections for more information.

A third possibility, although very rare, is that you are trying to set a tag with an invalid key. If the two solutions above don't resolve your problem, try reviewing all of the tags to be written to see if there are any that don't appear to be valid.

11.5 Picard just stopped working

There are typically two reasons that Picard will run very slowly or appear to be stalled:

Processing a large number of files at one time

When processing a large number of files in one batch, Picard can run into issues either due to processing each file (e.g.: AcoustID fingerprinting) or during lookups following clustering or fingerprinting because of all of the information requests to the MusicBrainz server API, as well as downloading cover art. Even though Picard may still be working its way through the backlog, the user interface may become non-responsive and appear that the program has stalled or frozen.

The impact of processing files in large batches is exacerbated when using plugins that make additional information request calls to the MusicBrains server API.

If you are processing a large library of files, it is generally more effective to process smaller batches (e.g.: 200 files) at a time, first retrieving the information using a cluster and lookup process, and then processing any remaining unmatched files using the scan process. Please see the *Retrieving Album information* section for more information.

Processing files across a network connection

If you are processing files across a network connection, this can impact the speed at which Picard works because of the speed difference between a network connection and a local drive. In this case, the throughput can be improved by first copying the source files to a local drive, process with Picard, and then move the resulting files to the network drive.

FREQUENTLY ASKED QUESTIONS

Some of the most often asked questions have been addressed in the following sections. These have been organized into groups based on the operation being performed.

12.1 Using Picard

How do I tag files with Picard?

There is a separate section that explains the tagging process. Please see *Using Picard* for details.

The green "Tagger" icon disappeared from MusicBrainz.org, how do I get it back?

This icon shows up when a manual lookup is performed via Picard using *"Tools → Lookup"*.

Alternatively the parameter `?tport=8000` can be added to the end of almost any MusicBrainz URL and the green tagger icons will continue to show up from then on.

I'm trying to load a release in Picard, but all I'm seeing is "Couldn't load album errors". What's up?

If you get "Couldn't load album errors" for releases in Picard, this can occur for a number of reasons. Check the following:

1. Is the problem persistent for a given release?

Try waiting a minute or two, or even a bit longer and then try again with a right-click, "Refresh". Sometimes the servers are just overloaded and temporarily reject requests.

2. Has the release been deleted from MusicBrainz?

If you are re-tagging files previously tagged with Picard, and get this error, the release has possibly been deleted. Try to right-click and use the "Lookup in browser" option to view the release on the website. If you can't find it, it may have been deleted. This could be because you tagged a pending release that was voted down, or tagged against a release that was deleted because editors decided it wasn't a valid release. This can happen for homebrew compilations, bad torrent or pirate rips, "advance" releases or very poorly added releases. Usually there will be an alternate release you can tag against, which you can find by searching or doing another clustered lookup from Picard. If you can't find a replacement and believe it has been

deleted unfairly, submit a new release[92], supplying evidence of the tracklisting and as much information as possible to prove it is genuine and it may be accepted again.

I'm using macOS, where are my network folders or drives?

These should show up in the add file and add folder dialogs, but they aren't visible by default in the file browser pane. If you want to see them in the file browser pane, right click in the pane and select "show hidden files". They should then be visible in the /Volumes folder.

12.2 File Formats

What formats does Picard support?

Picard supports the following file formats:

- MPEG-1 Audio (.mp3, .mp2, .m2a)
- MPEG-4 Audio (.m4a, .m4b, .m4p, .m4v, .mp4)
- Windows Media Audio (.wma, .wmv, .asf)
- Microsoft WAVE (.wav)
- The True Audio (.tta)
- FLAC (.flac)
- Audio Interchange File Format (.aiff, .aif, .aifc)
- Musepack (.mpc, .mp+)
- WavPack (.wv)
- OptimFROG (.ofr, .ofs)
- Monkey's Audio (.ape)
- Tom's lossless Audio Kompressor (.tak)
- Speex (.spx)
- Ogg FLAC (.oggflac)
- Ogg Theora (.ogg)
- Ogg Opus (.opus)
- Ogg Audio (.oga)
- Ogg Video (.ogv)
- Ogg Video (.ogv)
- ADTS stream / AAC (.aac)
- AC-3 (.ac3, .eac3)

---

[92] https://musicbrainz.org/doc/How_to_Add_a_Release

- Direct Stream Digital (.dsf)

Note: WAV files cannot be tagged due to the lack of a standard for doing so, however, they can be fingerprinted and renamed.

Note: The DFF type of Direct Stream Digital files cannot be tagged because they do not have the ability to hold metadata.

What formats will Picard support?

Picard is intended to eventually support all formats (including fingerprinting), but this is a complex (arguably never-ending) process, and will take some time.

Which tags can Picard write to my files?

See the *Tags & Variables* section for information on which MusicBrainz fields that Picard writes to tags. Picard Tag Mapping[93] contains more technical information on how these are further mapped into each tag format.

How to I edit several tags at once? Why is it not easier do so?

Please understand that Picard is not designed as a general purpose tag editor. Its primary goal is to retrieve community-maintained MusicBrainz data to write into your tags. Some secondary goals include:

- allowing rule-based customization of that data using scripts and plugins

- encouraging users to create an account and fix and update data via the MusicBrainz website, thus sharing their work with the rest of the community rather than simply fixing their tags locally.

To that end, Picard is likely to never have as much development focus on manual bulk editing of tags as other general purpose editors (e.g.: Mp3tag[94], foobar2000[95], or even many library managers such as iTunes, Windows Media Player, and MediaMonkey). That doesn't mean that the team won't welcome patches in this area!

Having said all this, it is still possible to edit several tags at once in Picard by following the steps:

1. Click and select several files with CTRL or SHIFT

2. Right click on one of them, then click Details...

3. On the popup dialog you can see the tags, with entries that denote where tags are different across files. You can edit or add new tags here.

4. On exiting the dialog, you have changed the tags in memory. You need to click Save in order to persist these changes to your files.

---

[93] https://picard.musicbrainz.org/docs/mappings/
[94] https://www.mp3tag.de/en/
[95] https://www.foobar2000.org/

This process should work in both panes.

The built-in audio player cannot play my file. Which formats does it support?

The formats supported by the built-in audio player depend on the formats supported by your operating system.

Windows:

The supported formats depend on the installed codecs. Depending on the Windows version certain codecs are pre-installed, but you can install additional codecs.

You might want to install the Directshow Filters for Ogg[96] to add support for Ogg Vorbis, Ogg Speex, Ogg Theora, Ogg FLAC, native FLAC, and WebM files.

See also:

Additional information is available from Microsoft's Codecs FAQ[97].

Linux:

On Linux systems the player uses GStreamer which supports most common audio formats, although some distributions might exclude some codecs due to licensing issues. For the widest format support make sure you install all of the GStreamer plugins available for your distribution.

I am using Fedora. Why doesn't acoustic fingerprinting work?

Acoustic fingerprinting in Picard uses a tool called `fpcalc`, which is not available in Fedora. You can get it by installing the chromaprint-toolspackage from the RPM Fusion repository[98]. This functionality is not contained in the main Fedora `picard` package because it requires the `ffmpeg` package which cannot be distributed by Fedora[99]. After enabling the "rpmfusion-free" RPM Fusion repository[100], install the package using (as root):

```
yum install chromaprint-tools
```

12.3 Configuration

Where is the Picard configuration saved?

Picard saves the configuration in the file `Picard.ini`. Its location depends on the operating system:

Windows:

```
%APPDATA%\MusicBrainz\Picard.ini
```

---

[96] https://xiph.org/dshow/downloads/
[97] https://support.microsoft.com/en-us/help/15070/windows-media-player-codecs-frequently-asked-questions
[98] https://rpmfusion.org/
[99] https://fedoraproject.org/wiki/Forbidden_items
[100] https://rpmfusion.org/Configuration

This usually will be `C:\Users\YourUserName\AppData\Roaming\MusicBrainz`, where `YourUserName` should be replaced with your actual Windows user name.

macOS, Linux and other Unix like systems:

`$HOME/.config/MusicBrainz/Picard.ini`

I tagged a file in Picard, but iTunes is not seeing the tags!

First, you need to force iTunes to re-read the information from your tags and update its library. This is discussed in the iTunes Guide[101].

Additionally, iTunes has a known bug in its ID3v2.4 implementation, which makes it unable to read such tags if they also contain embedded cover art. As a work-around, you can configure Picard to write ID3v2.3 tags.

My tags are truncated to 30 characters in Windows Media Player!

Prior to version 0.14, Picard's default settings were to write ID3v2.4 and ID3v1 tags to files. WMP can't read ID3v2.4, so it falls back to ID3v1 which has a limitation of 30 characters per title. To solve this on versions prior to 0.14, configure Picard to write ID3v2.3 tags instead.

Starting with version 0.14, the default settings have been changed to ID3v2.3 and this should no longer be an issue.

How do I tell Picard which browser to use?

On Windows, macOS, GNOME and KDE, Picard uses the default browser that has been configured for the system. On other systems, you can use the `BROWSER` environment variable.

For example:

```
export BROWSER="firefox '%s' &"
```

Another approach that works in some GNU/Linux systems is the following command:

```
sudo update-alternatives --config x-www-browser
```

This should present you with a list of existing browsers in your system, allowing you to select the one to use by default.

---

[101] https://musicbrainz.org/doc/iTunes_Guide

EXAMPLES

This section is still under development.

CHAPTER

FOURTEEN

APPENDICES

14.1 Plugins API

14.1.1 Plugin Metadata

Each plugin must provide some metadata as variables. Those variables should be placed at the top of the file.

```
PLUGIN_NAME = "Example plugin"
PLUGIN_AUTHOR = "This authors name"
PLUGIN_DESCRIPTION = "This plugin is an example"
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.1', '2.2']
PLUGIN_LICENSE = "GPL-2.0-or-later"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.html"
```

Variables explanation:

- **PLUGIN_DESCRIPTION** should be as simple as possible, while still describing the main function.

- **PLUGIN_VERSION** should be filled with the version of Plugin.

- **PLUGIN_API_VERSIONS** should be set to the versions of Picard this plugin to run with. New Picard versions will usually support older plugin API versions, but on breaking changes support for older plugin versions can be dropped. Versions available for Picard 2 are "2.0", "2.1" and "2.2".

- **PLUGIN_LICENSE** should be set with the license name of the plugin. If possible use one of the license names from the SPDX License List[102], but you are welcomed to use another licnese if the one you chose is not available in the list.

- **PLUGIN_LICENSE_URL** should be set to a URL pointing to the full license text.

---
[102] https://spdx.org/licenses/

14.1.2 Metadata Processors

MusicBrainz metadata can be post-processed at two levels, album and track. The types of the arguments passed to the processor functions in the following examples are as follows:

- **album**: `picard.album.Album`

- **metadata**: `picard.metadata.Metadata`

- **release**: `dict` with release data from MusicBrainz JSON web service

- **track**: `dict` with track data from MusicBrainz JSON web service

Album metadata example:

```python
PLUGIN_NAME = "Disc Numbers"
PLUGIN_AUTHOR = "Lukas Lalinsky"
PLUGIN_DESCRIPTION = "Moves disc numbers from album titles to tags."

from picard.metadata import register_album_metadata_processor
import re

def remove_discnumbers(tagger, metadata, release):
    matches = re.search(r"\(disc (\d+)\)", metadata["album"])
    if matches:
        metadata["discnumber"] = matches.group(1)
        metadata["album"] = re.sub(r"\(disc \d+\)", "", metadata["album"])

register_album_metadata_processor(remove_discnumbers)
```

Track metadata example:

```python
PLUGIN_NAME = "Feat. Artists"
PLUGIN_AUTHOR = "Lukas Lalinsky"
PLUGIN_DESCRIPTION = "Removes feat. artists from track titles."

from picard.metadata import register_track_metadata_processor
import re

def remove_featartists(tagger, metadata, track, release):
    metadata["title"] = re.sub(r"\(feat. [^)]*\)", "", metadata["title"])

register_track_metadata_processor(remove_featartists)
```

14.1.3 Event Hooks

Plugins can register themselves to listen for different events. Currently the following event hooks are available:

file_post_load_processor(file)

This hook is called after a file has been loaded into Picard. This could for example be used to load additional data for a file. Usage:

```python
from picard.file import register_file_post_load_processor

def file_post_load_processor(file):
  pass

register_file_post_load_processor(file_post_load_processor)
```

file_post_save_processor(file)

This hook is called after a file has been saved. This can for example be used to run additional post-processing on the file or write extra data. Note that the file's metadata is already the newly saved metadata. Usage:

```python
from picard.file import register_file_post_save_processor

def file_post_save_processor(file):
  pass

register_file_post_save_processor(file_post_save_processor)
```

file_post_addition_to_track_processor(track, file)

This hook is called after a file has been added to a track (on the right-hand pane of Picard).

```python
from picard.file import register_file_post_addition_to_track_processor

def file_post_addition_to_track_processor(track, file):
  pass

register_file_post_addition_to_track_processor(file_post_addition_to_track_
 ↪processor)
```

file_post_removal_from_track_processor(track, file)

This hook is called after a file has been removed from a track (on the right-hand pane of Picard).

```python
from picard.file import register_file_post_removal_from_track_processor

def file_post_removal_from_track_processor(track, file):
  pass

register_file_post_removal_from_track_processor(file_post_removal_from_track_
↪processor)
```

album_post_removal_processor(album)

This hook is called after an album has been removed from Picard.

```python
from picard.album import register_album_post_removal_processor

def album_post_removal_processor(album):
  pass

register_album_post_removal_processor(album_post_removal_processor)
```

Note: Event hooks have been available since API version 2.2.

14.1.4 File Formats

Plugins can extend Picard with support for additional file formats. See the existing file format implementations[103] for details on how to implement the _load and _save methods. Example:

```python
PLUGIN_NAME = "..."
PLUGIN_AUTHOR = "..."
PLUGIN_DESCRIPTION = "..."
PLUGIN_VERSION = '...'
PLUGIN_API_VERSIONS = ['...']
PLUGIN_LICENSE = "..."
PLUGIN_LICENSE_URL = "..."

from picard.file import File
from picard.formats import register_format
from picard.metadata import Metadata

class MyFile(File):
    EXTENSIONS = [".foo"]
    NAME = "Foo Audio"
```

(continues on next page)

---

[103] https://github.com/metabrainz/picard/tree/master/picard/formats

---

```
    def _load(self, filename):
        metadata = Metadata()
        # Implement loading and parsing the file here.
        # This method is supposed to return a Metadata instance filled
        # with all the metadata read from the file.
        metadata['~format'] = self.NAME
        return metadata

    def _save(self, filename, metadata):
        # Implement saving the metadata to the file here.
        pass

register_format(MyFile)
```

14.1.5 Tagger Script Functions

To define new tagger script functions use `register_script_function(function, name=None)` from the `picard.script` module. `parser` is an instance of `picard.script.ScriptParser`, and the rest of the arguments passed to it are the arguments from the function call in the tagger script. Example:

```
PLUGIN_NAME = "Initials"
PLUGIN_AUTHOR = "Lukas Lalinsky"
PLUGIN_DESCRIPTION = "Provides tagger script function $initials(text)."
PLUGIN_VERSION = '0.1'
PLUGIN_API_VERSIONS = ['2.0']
PLUGIN_LICENSE = "GPL-2.0"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.txt"

from picard.script import register_script_function

def initials(parser, text):
    return "".join(a[:1] for a in text.split(" ") if a[:1].isalpha())

register_script_function(initials)
```

`register_script_function` supports two optional arguments:

- **eval_args**: If this is **False**, the arguments will not be evaluated before being passed to **function**.

- **check_argcount**: If this is **False** the number of arguments passed to the function will not be verified.

The default value for both arguments is **True**.

14.1.6 Context Menu Actions

Right-click context menu actions can be added to albums, tracks and files in "Unmatched Files", "Clusters" and the "ClusterList" (parent folder of Clusters). Example:

```
PLUGIN_NAME = u'Remove Perfect Albums'
PLUGIN_AUTHOR = u'ichneumon, hrglgrmpf'
PLUGIN_DESCRIPTION = u'''Remove all perfectly matched albums from the
↪selection.'''
PLUGIN_VERSION = '0.2'
PLUGIN_API_VERSIONS = ['0.15.1']
PLUGIN_LICENSE = "GPL-2.0"
PLUGIN_LICENSE_URL = "https://www.gnu.org/licenses/gpl-2.0.txt"

from picard.album import Album
from picard.ui.itemviews import BaseAction, register_album_action

class RemovePerfectAlbums(BaseAction):
    NAME = 'Remove perfect albums'

    def callback(self, objs):
        for album in objs:
            if isinstance(album, Album) and album.is_complete()\
                and album.get_num_unmatched_files() == 0\
                and album.get_num_matched_tracks() == len(list(album.
↪iterfiles())))\
                and album.get_num_unsaved_files() == 0 and album.loaded ==
↪True:
                self.tagger.remove_album(album)

register_album_action(RemovePerfectAlbums())
```

Use `register_x_action` where 'x' is "*album*", "*track*", "*file*", "*cluster*" or "*clusterlist*".